DK

ScreenShot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM

IAN GRAHAM

BOOK ONE

A must for every beginner – the first full-colour guide to easy programming

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM

THE DK SCREEN-SHOT PROGRAMMING SERIES

Never has there been a more urgent need for a series of well-produced, straightforward, practical guides to learning to use a computer. It is in response to this demand that The DK Screen-Shot Programming Series has been created. It is a completely new concept in the field of teach-yourself computing. And it is the first comprehensive library of highly illustrated, machine-specific, step-by-step programming manuals.

BOOKS ABOUT THE ZX SPECTRUM

This is Book One in a series of unique step-by-step guides to programming the ZX Spectrum. Together with its companion volumes, it will build up into a self-contained teaching course that begins with the basic principles of programming, and progresses – via more sophisticated techniques and routines – to an advanced level.

BOOKS ABOUT OTHER COMPUTERS

Additional titles in the series will cover each of the world's most popular computers. These will include:

Step-by-Step Programming for the **BBC Micro**

Step-by-Step Programming for the **Commodore 64**

Step-by-Step Programming for the **Acorn Electron**

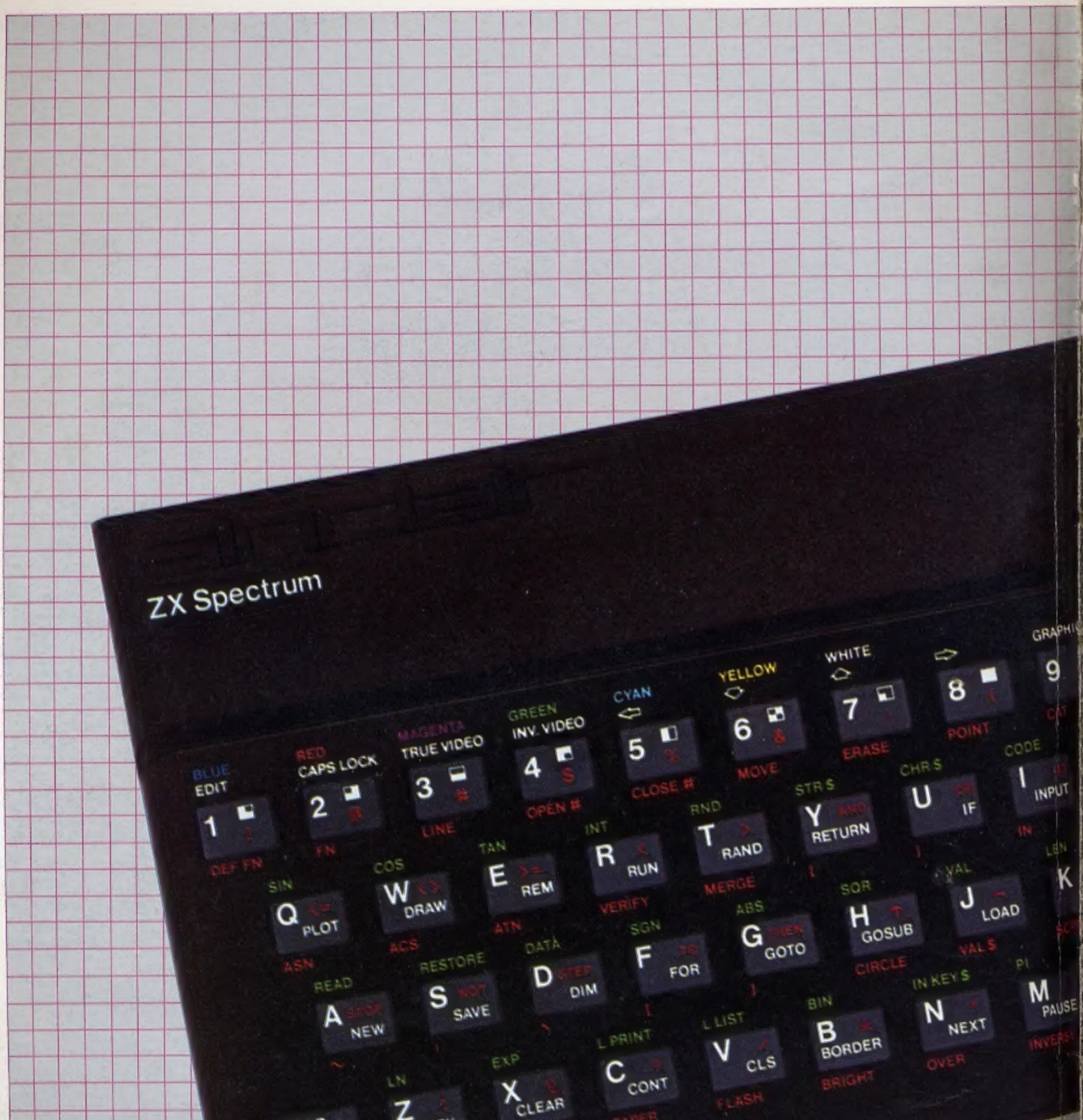
Step-by-Step Programming for the **Apple II**

Step-by-Step Programming for the **IBM PCjr**

IAN GRAHAM

After taking a B.Sc. in Applied Physics and a postgraduate diploma in Journalism at The City University, London, Ian Graham worked as assistant editor of *Electronics Today International* and deputy editor of *Which Video?* Since becoming a full-time freelance writer in 1982, he has contributed to a wide range of technical magazines (including *Computing Today, Video Today, Video Search, Hobby Electronics, Electronic Insight, Popular Hi-Fi, Science Now,* and *Next...)* and has also written a number of popular books on computers and computing. These include *Computer & Video Games, Information Technology, The Inside Story – Computers,* and *The Personal Computer Handbook* (co-written with Helen Varley).

BOOK ONE


ZX Spectrum
BLUE EDIT 1 DEF FN
RED CAPS LOCK 2 FN
MAGENTA TRUE VIDEO 3 LINE
GREEN INV. VIDEO 4 OPEN #
CYAN 5 CLOSE #
YELLOW 6 MOVE
WHITE 7 ERASE
8 POINT
9 CAT
CODE I INPUT
CHR$ U IF
STR$ Y RETURN
RND T RAND
INT R RUN
TAN E REM
COS W DRAW
SIN Q PLOT
ASN
ACS
ATN
VERIFY
MERGE
ABS G GOTO
SGN F FOR
DATA D DIM
RESTORE S SAVE
READ A NEW
L PRINT C CONT
L LIST V CLS
BIN B BORDER
SQR H GOSUB
CIRCLE
VAL J LOAD
VAL$
IN KEY$ N NEXT
OVER
PI M PAUSE
INVERSE
LEN K
SCR
LN Z COPY
BEEP
EXP X CLEAR
INK
PAPER
FLASH
BRIGHT
CAPS SHIFT

DK

ScreenShot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM



IAN GRAHAM

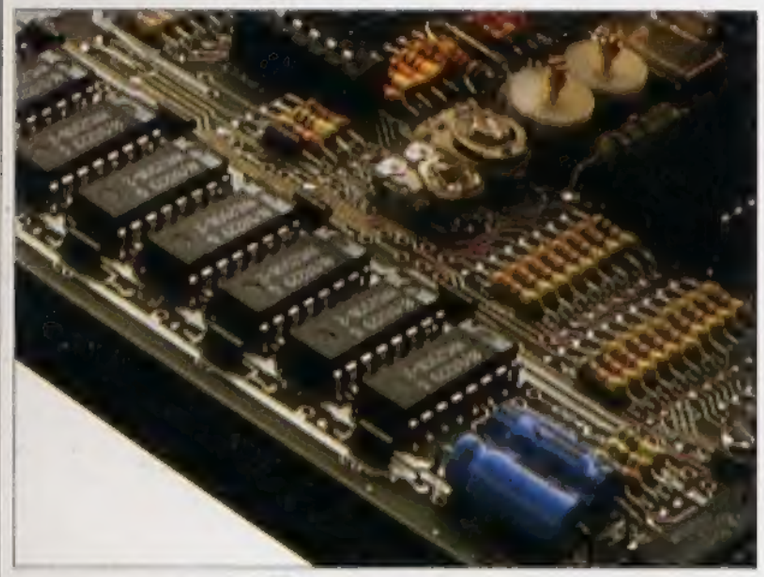
DORLING KINDERSLEY·LONDON

BOOK ONE

CONTENTS

6 THE ZX SPECTRUM

8 INSIDE THE COMPUTER



10 THE SPECTRUM KEYBOARD

12 SETTING UP



14 USING THE SCREEN

16 COMPUTER CALCULATIONS

18 WRITING YOUR FIRST PROGRAM

20 DISPLAYING YOUR PROGRAMS

```
  10 CLS
  20 INPUT "What is your name";n
$
  30 PRINT "**********************
**********"
  40 PRINT "ZX Spectrum programm
ed by ";n$
  50 PRINT "**********************
**********"
  30>PRINT "**********************
**********"
  40 PRINT "ZX Spectrum programm
ed by ";n$
  50 PRINT "**********************
**********"
```

22 CORRECTING MISTAKES

24 COMPUTER CONVERSATIONS

26 WRITING PROGRAM LOOPS

28 THE ELECTRONIC DRAWING-BOARD

```
 10 CLS
 20 PLOT 93,16
 30 DRAW 160,88
 40 DRAW -176,0
 50 DRAW 160,-88
 60 DRAW -72,136
 70 DRAW -72,-136
```


The DK Screen-Shot Programming Series was conceived, edited and designed by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.

Project Editor David Burnie
Art Editor Peter Luff
Design Assistant Steve Wilson
Photography Vincent Oliver
Managing Editor Alan Buckingham
Art Director Stuart Jackman

First published in Great Britain in 1984 by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.
Second impression 1984

Copyright © 1984 by Dorling Kindersley Limited, London
Text copyright © 1984 by Ian Graham

As used in this book, any or all of the terms SINCLAIR, ZX SPECTRUM, ZX MICRODRIVE, MICRODRIVE CARTRIDGE, and ZX PRINTER are Trade Marks of Sinclair Research Limited.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright owner.

British Library Cataloguing in Publication Data

Graham, Ian, 1953–
Step by step programming for the ZX Spectrum. Book 1.
1. Sinclair ZX Spectrum (Computer)—— Programming
I. Title
001.64'2 QA76.8.S625

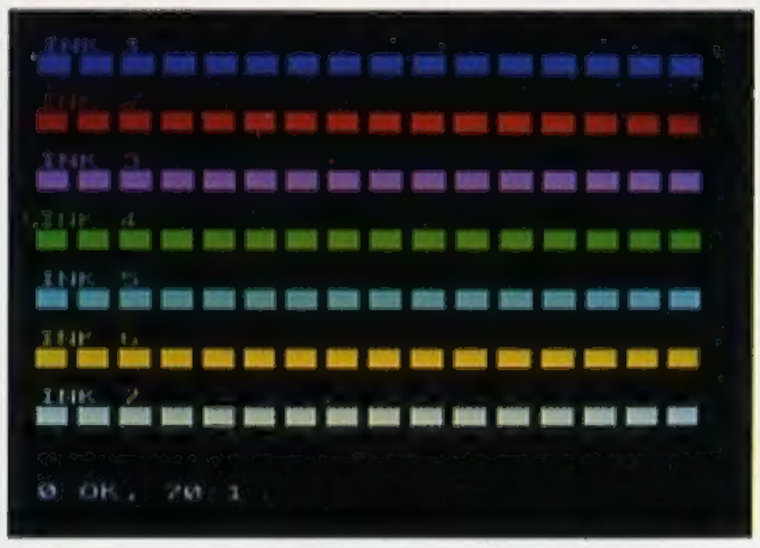
ISBN 0-86318-026-4

Typesetting by The Letter Box Company (Woking) Limited, Woking, Surrey, England
Reproduction by Reprocolor Llovet S.A., Barcelona, Spain
Printed and bound in Italy by A. Mondadori, Verona

30
DESIGNING YOUR OWN CHARACTERS

32
ANIMATION

34
INTRODUCING COLOUR



36
COLOUR GRAPHICS



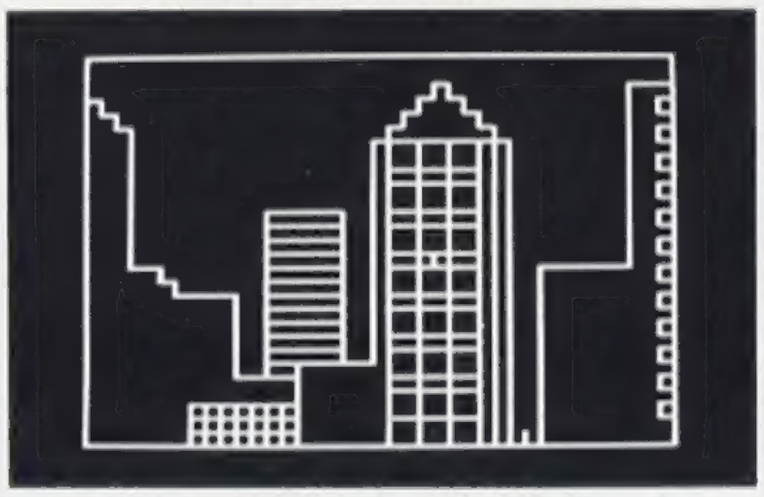
38
SPECIAL SCREEN TECHNIQUES 1

40
SPECIAL SCREEN TECHNIQUES 2

42
SOUND, NOTES AND MUSIC

44
SPECIAL EFFECTS WITH SOUND

46
DECISION-POINT PROGRAMMING



48
UNPREDICTABLE PROGRAMS

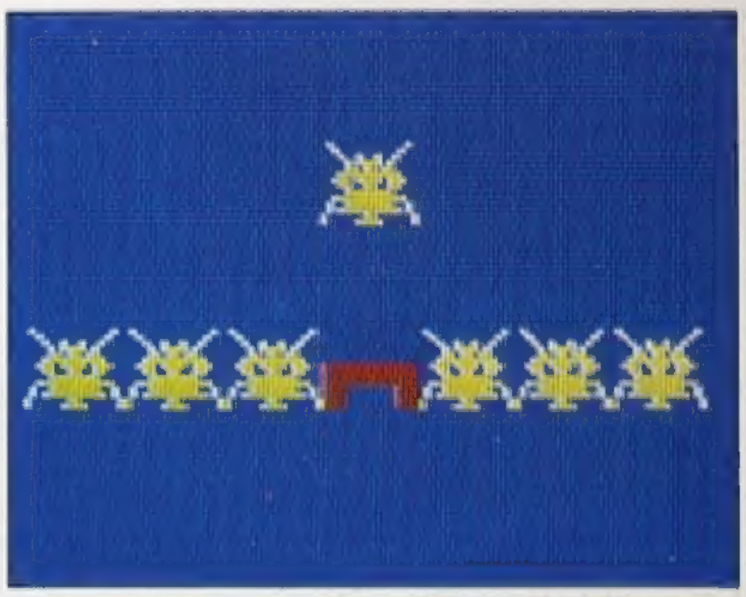
50
COMPILING A DATA BANK

52
QUICK WAYS TO STORE CHARACTERS

54
ADVANCED COLOUR GRAPHICS



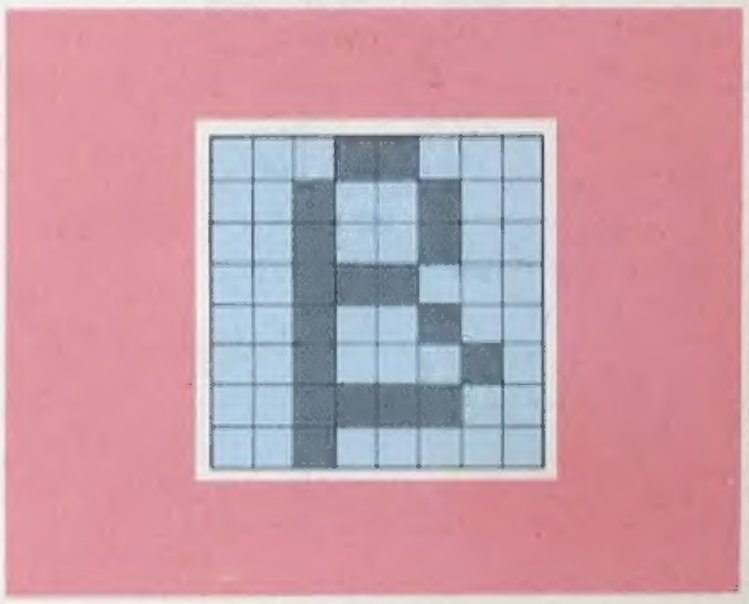
56
WRITING SUBROUTINES



58
HINTS AND TIPS

60
HOW TO KEEP YOUR PROGRAMS

61
GRAPHICS AND CHARACTER GRIDS



62
GLOSSARY

64
INDEX

THE ZX SPECTRUM

Since its launch in April 1982, the ZX Spectrum has become one of the most popular home computers available today. Despite its small size, it is potentially very powerful. It contains the same basic components as much larger computers and uses its own version or "dialect" of the most popular home computer language, BASIC. The Spectrum offers an inexpensive way of learning to program a computer with many of the features found in larger, more elaborate systems.

Two versions of the Spectrum are available. They are identical but for the size of the memory – the 48K model can hold more information than the 16K version. The 48K model has a memory capacity which is greater than many more expensive personal computers. All the programs in this book will work equally well on either the 16K or 48K machine.

Connectors and peripherals

From the outside the Spectrum seems to be little more than a slim black case with a keyboard on top, which is composed of soft moving keys. Although it looks like a typewriter keyboard, it is actually very different. As you can see on pages 10–11, each key can produce whole words as well as letters, and when used in programming, each key can perform up to six different functions.

If you turn the computer around so that you are looking at its back panel, you will see that there are four sockets and a slot. The small sockets connect the Spectrum to a television, cassette recorder and power supply. You can find out how to use a cassette recorder with the computer on page 60. The longer slot is the edge connector, which is actually a part of the Spectrum's circuit board exposed at the edge of the computer casing. The metallic strips on the edge of the board are used to connect extra pieces of hardware, such as printers, microdrives or joysticks, to the computer. When you are handling the computer, do not touch any of these contacts, as grease and dirt can cause mulfunctions if you later use the edge connector with any "peripherals".

The Spectrum produces a colour television picture, but it can also be viewed in black and white. In this case, the different colours show up as shades of grey. The computer itself produces all the sound effects used in programs. If you turn the Spectrum over so that you are looking at the bottom panel, you will be able to see the circle of holes in one corner that is the sound outlet for the Spectrum's tiny loudspeaker. It is capable of beeping or playing simple tunes.



9 Volt DC socket The Spectrum is supplied with a power adaptor that transforms the high-voltage alternating current (AC) supply into a 9 Volt direct current (DC) supply suitable for the computer.

Edge connector This multi-terminal socket connects the computer with a range of hardware including the Spectrum printer, Microdrive units and "analogue" controls such as games joysticks.

MIC socket When this is connected to a cassette recorder's microphone socket, programs can be transferred from the computer to be stored on cassette.

EAR socket This socket allows the Spectrum to receive stored programs from a tape cassette. It is connected to the cassette recorder's EAR socket.

TV socket The picture signals that the Spectrum produces are fed into a television's aerial socket from the Spectrum's TV socket, using a cable supplied with the computer.





INSIDE THE COMPUTER

The ZX Spectrum is constructed on a single printed circuit board. The major components are integrated circuits, or "chips", which look like thin rectangular slices of black plastic with up to 20 metal pins protruding from each edge. These connect the chips to contacts which run over the surface of the board.

At the heart of the Spectrum is a microprocessor which makes up the computer's Central Processing Unit (CPU). The CPU does all the computer's calculations, monitors the keyboard and acts on any key-presses it detects. But despite the CPU's complexity, it can only follow instructions that it is given. The instructions you type on the keyboard must first be translated into the numerical machine code that the computer works in.

The program required to perform this translation is stored permanently in a Read Only Memory (ROM), also known as a "non-volatile" memory. This memory is not free for you to store your own programs in – its contents can only be read. The program is unaffected by the computer being turned on or off.

The Spectrum's Random Access Memory (RAM) offers storage space for the user. Everything you type in on the keyboard is stored in RAM until you either type in the keyword NEW or switch off the power. Because the contents of RAM are erased when the power is interrupted, it is also called a "volatile" memory.

Main board components

The two versions of the Spectrum are distinguished by the size of their RAMs – 16K and 48K. The K stands for kilobytes, each equivalent to 1024 bytes. A byte is a standard piece of electronic information in binary form. It is made up of eight bits – pulses of electricity which each represent a 0 or 1. Bytes can be thought of as acting like words in the computer system, with bits being the letters. The main difference between the Spectrum's system of communication and English is that computer words are all eight letters long, and are made up from an alphabet containing only two letters. The Spectrum does everything from producing colour television pictures to playing tunes by using this binary code.

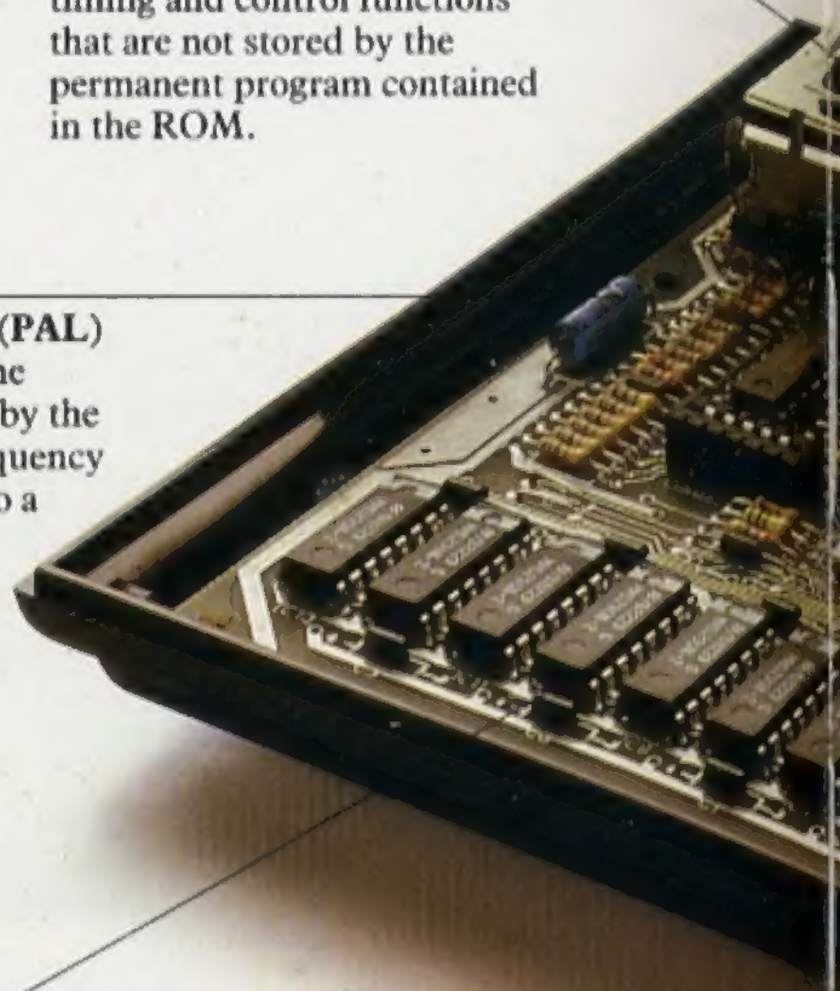
All the computer's activities are synchronized so that the correct information is available in the right place at the right time. This synchronization is achieved by an internal clock. The clock is based on a crystal oscillator that "ticks" at the rate of 3.5 million pulses per second. Additional timing and control operations are provided by a large chip called an Uncommitted Logic Array (ULA), which carries out complicated logic functions.

The microchip command chain All the chips within the Spectrum form an electronic chain of command, with the CPU performing all the executive tasks. The rest of the chips – including the RAMs, ROM and ULA – act as temporary or permanent information storage systems. These supply the CPU with the instructions it needs to carry out the computer's functions.

In this view of the computer's interior, the connectors that link the keyboard to the rest of the computer have been detached. It is advisable not to open your Spectrum as these connectors can easily be broken.

Uncommitted Logic Array (ULA) This provides additional timing and control functions that are not stored by the permanent program contained in the ROM.

Phase Alternation Line (PAL) encoder This converts the stream of data produced by the Spectrum into a high-frequency signal that can be fed into a television.



Random Access Memory (RAM) Eight RAM chips provide 16K of storage for all the programming information that the computer is given after being switched on. The second group of eight further RAM chips provides an additional 32K of memory in the 48K version of the Spectrum.



Cassette recorder sockets These are used to record or play back programs.

Logic chips This area deals with a variety of logic functions including those needed to "interface" with hardware linked to the computer.

Central Processing Unit (CPU) The executive part of the computer. This microprocessor carries out all calculations and controls activities in the rest of the computer, drawing on information held in both ROM and RAM.

Read Only Memory (ROM) This chip contains the instructions necessary to turn programs into a form that the computer's most important chip, the CPU, can understand. Unlike RAM, its contents do not disappear when the power is switched off. It contains the computer's BASIC interpreter.

9 Volt DC socket The power supply socket.

Voltage regulator This prevents changes in voltage disrupting the activities of the computer.

Loudspeaker The speaker produces notes and sound effects when called on by a program.

THE SPECTRUM KEYBOARD

On a conventional typewriter keyboard, each key is used to print a lower case letter and (with the shift key) the upper case version of the same letter. The Spectrum keyboard works like this, but because it has more than one shift key, it is much more versatile. The keys on the Spectrum are capable of selecting as many as six different functions. The 40 Spectrum keys can produce a total of around 200 letters, words and symbols. You may find using the keyboard a slow process to begin with, but once you have mastered the use of the shift keys, finding your way around all the different words and symbols will soon become second nature. Two different kinds of shift key – CAPS SHIFT and SYMBOL SHIFT – are used either independently or together to select key functions.

Keyboard technique

The Spectrum keys are moving, calculator-style buttons. Every time one is pressed, making a character or word appear on the television screen, the computer produces a single click to let you know that the contact has been made. If you hold a key down for more than a couple of seconds, the symbol is repeatedly printed.

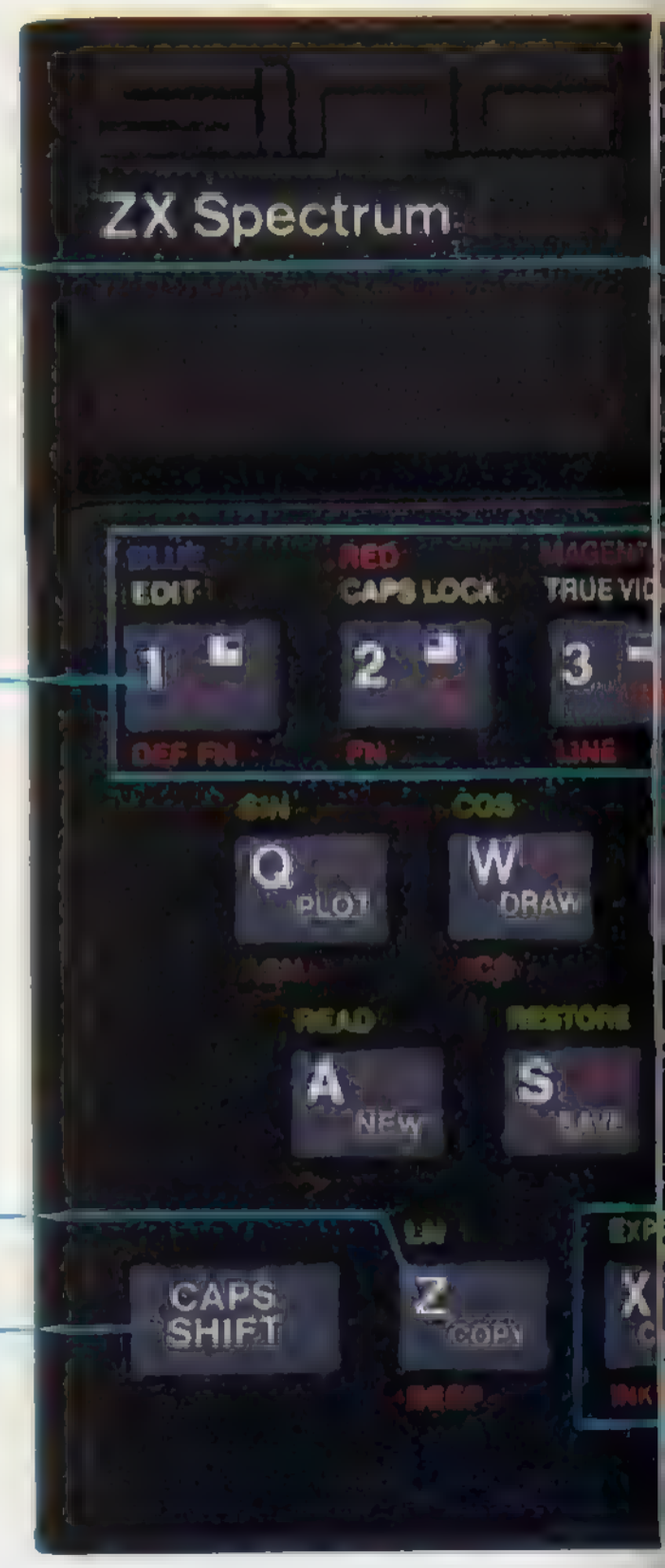
You don't have to be an accomplished typist to use the keyboard. The Spectrum saves you a lot of typing because the command words used by the computer need not be typed out in full. Pressing a key once makes the whole word printed on the key appear on the television screen. To help you further, commands and symbols that are often used together in programs are, wherever possible, grouped on adjacent keys.

Understanding the cursor

The Spectrum accepts instructions only if they are in a logical sequence, and it actually tells you what kind of instruction or symbol it expects by showing one of five flashing cursors. If the cursor is a "K", the computer is expecting you to supply a keyword next. This is simply any of the command words in white on the keyboard. So, if you press the P key, the word PRINT, and not the letter P, appears on the screen. If the cursor is flashing "L", pressing the P key produces a lower case letter p on the screen. To produce a capital P, press the CAPS SHIFT key while pressing p. This will make the cursor change briefly from "L" to "C" to show that capitals have been selected. The "G" cursor will appear when you use the GRAPHICS key, while the "E" (extended mode) cursor will appear when you use the shift keys to select keywords printed on the keyboard in red or green. It reverts to "L" after ENTER is pressed.

Number keys These offer a quick and easy way of producing graphics when they are used after pressing CAPS SHIFT with GRAPHICS. The graphics symbol on the number key will then appear on the screen. Keys 0 to 7 also control the colours produced on the screen.

EDIT This is used to "extract" a line from a program in order to change or edit it. The EDIT function is selected by the CAPS SHIFT key.



BEEP When the shift keys are used to select the extended mode, this key programs the command which controls the Spectrum's sound synthesizer.

CAPS SHIFT This allows you to select the upper case (capital) version of a letter, instead of the lower case version normally used. It is also used with SYMBOL SHIFT to obtain the words and characters above and below the keys.

Screen display keys The red keywords under keys X to M produce the commands controlling the way the text and screen background is displayed. The keywords INK and PAPER, together with the white keyword BORDER, are used in conjunction with the colour keys.

Cursor controls These four keys are used to direct the screen cursor to any point in a program that needs alteration. The cursor function is selected by CAPS SHIFT.

GRAPHICS When used with CAPS SHIFT, this key switches the Spectrum to the graphics mode, ready to accept graphics symbols from keys 1 to 8.

DELETE When used with CAPS SHIFT, this key backspaces the cursor and deletes keywords and symbols on the screen. It also codes for the "colour" black.

PRINT This key carries the frequently-used combination PRINT". Pressing this key once produces PRINT, while pressing it again while pressing SYMBOL SHIFT produces the double quotation marks.

ENTER The Spectrum will not respond to most commands unless they are followed by this key. It is roughly equivalent to the typewriter's carriage return key. When the ENTER key is pressed, the Spectrum will then respond to a command or point out any mistakes in typing.

SPACE This is equivalent to the typewriter's space bar. It also has a second function – to BREAK or halt a program before it has finished running. BREAK is selected by CAPS SHIFT.

SYMBOL SHIFT The red symbol on each key is selected by holding this key down while the key required is pressed. It may also be used in combination with CAPS SHIFT.

How to select key functions

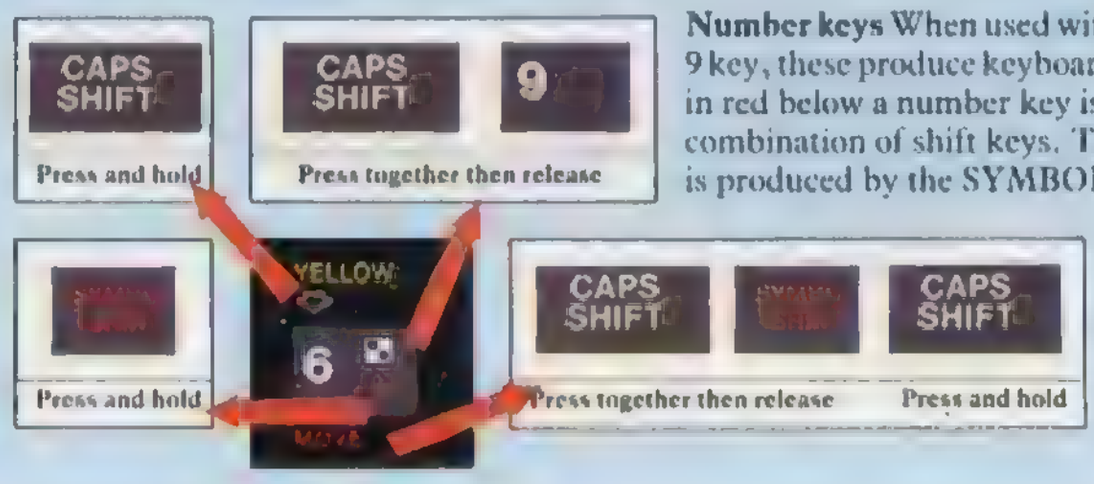
Letter keys When used with the "K" cursor, a letter key will produce a keyword. The cursor will then change to "L", and the letter key will then produce letters. The keywords above and below the key are produced by combinations of the shift keys.



Unshifted functions If no shift keys are used, a letter key will produce first its white keyword, then the lower case letter.



Number keys When used with CAPS SHIFT and the 9 key, these produce keyboard graphics. The keyword in red below a number key is produced by a combination of shift keys. The red symbol on the key is produced by the SYMBOL SHIFT key.



Unshifted functions If no shift keys are used, a number key will just produce numbers.

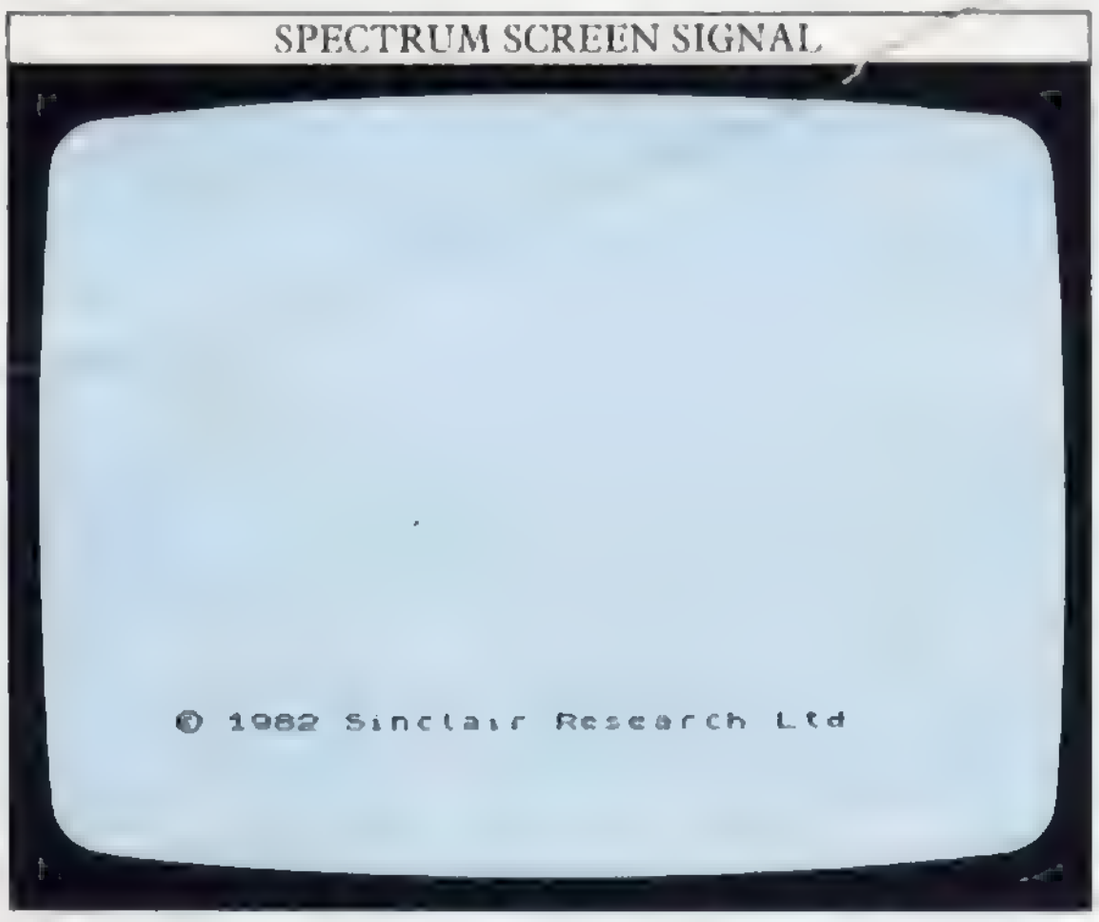
SETTING UP

Setting up your Spectrum is quite straightforward and logical. Getting the best possible results on the television screen sometimes takes a little longer. To begin with, connect your Spectrum power supply to the mains and link it to the computer. The computer has no on/off switch. As soon as you connect it to the power supply, you should be able to hear the computer humming if the connection has been made.

Now you need to make a connection between the Spectrum and the television so that you can see the results of your programming. Take the black lead supplied with the computer and plug it into the television's aerial socket. Plug the other end into the Spectrum socket marked "TV", and then switch on the power to the television.

The first results that you see will probably look like a blizzard, accompanied by a loud hissing coming from the television loudspeaker. Turn the television volume control down as far as it will go. The Spectrum will produce all the sound effects you program with its own built-in loudspeaker. Now switch the television to a channel that you can allocate permanently to the computer. The television treats the Spectrum's signal like any ordinary broadcast, so you have to tune the television just as you would to watch a television program. Adjust the tuner controls until you see this on the screen:

SPECTRUM SCREEN SIGNAL



If you cannot get any picture at all from the computer, check that all the power connections have been made properly. Next, check that the Spectrum's TV socket is connected to the television's aerial socket and make sure that the channel you are tuning is the one selected. You should only have to tune your television once. After that just selecting the right channel should produce a correctly tuned display.

How to test the Spectrum's colours

To enable you to test all the Spectrum's colours, you can use this very simple series of commands to see all the colours on screen. Having turned the computer on, press the following sequence of keys (ENTER here indicates the ENTER key at the right of the keyboard):

```
B1 ENTER
B2 ENTER
B3 ENTER
B4 ENTER
B5 ENTER
B6 ENTER
B7 ENTER
```

Every time you press the ENTER key, you should see a change in colour in the "border" area around the screen. (The techniques used to produce colours are explained on pages 34–35). B1 ENTER should colour the screen like this:

B1 COLOUR DISPLAY



When you use this sequence of colour commands to change the colour of the screen, you may find that they seem to have no effect. The tuning required to pick up the Spectrum's colour signal precisely is quite delicate, and you will need to experiment for a while to get the best results. If you do get a picture, but then if you cannot produce colour on the television, there is a possibility that your television is not able to interpret the Spectrum's colour signals. Your dealer should be able to advise you if this is the case.

Although the Spectrum's output is normally viewed on an ordinary television receiver, the signal can be fed into another type of television set known as a monitor. This contains everything that your television receiver has except a tuner, so it cannot receive television broadcasts. Changing the Spectrum's stream of data to

a high-frequency television signal and then reversing the process inside the television reduces the picture quality. By eliminating these stages, a monitor is able to produce better quality pictures. The screen photographs in this book were taken using a monitor, so your own television may produce slightly less clear displays.

Connecting peripherals

The next connection you will want to make is to a tape cassette recorder. The method for using a tape recorder to save your programs is covered in detail on page 60. If you do want to use the cassette recorder, make sure that you have the connecting lead. This is a two-core flex which is coloured black and grey. It is very important that you do not cross-connect this, or the cassette storage and playback will not work.

Later, you may wish to add a printer to your system. The Sinclair ZX printer is supplied with a short cable terminated by a plug which clamps onto the edge connector in the Spectrum's rear panel. It cannot be fitted the wrong way round. Microdrive units are fitted through an interface which sits underneath the computer, again linking up with the edge connector.

When you are using the edge connector, don't force a plug in if you feel any resistance. You may be pushing the plug in wrongly, and this could cause damage.



Arranging the computer To make using the computer as comfortable as possible, the television screen should be lined up behind the computer so that both can be seen without turning your head. The screen should not be too close.

Cassettes used for program storage should be kept away from the power supply, computer and television. Recorded programs may otherwise be disrupted by the magnetic fields produced by these pieces of equipment.

USING THE SCREEN

Having set up your Spectrum, you may already have given in to the temptation to tap a few keys and see what happens. If not, try it – you can't do any damage. The first thing you will notice is that the Sinclair copyright line at the bottom of the screen disappears, and is replaced by a line of characters. But as you will have seen on pages 10–11, exactly which characters appear depends not only on which keys you press, but also on the combination of keys that you use.

Starting to PRINT

To make some sense of this apparently confusing situation, disconnect the power for a second or two to clear the computer's memory and then press the key with PRINT on it (the P key) followed by one of the number keys, and then by the key marked ENTER. As soon as you press the ENTER key, the number you pressed will appear at the top of the screen. You can use PRINT to put a series of numbers on the screen:

PRINT WITH NUMBERS



If, instead of disconnecting the power, you press the key marked CLS (the V key) and then the ENTER key, everything that you have PRINTed will disappear.

What is a variable?

Now try typing in:

PRINT x

followed by the ENTER key. The computer will respond to this – or a command to PRINT any other letter – by displaying an error report:

2 Variable not found, 0:1

This report, one of many that the computer has stored in its permanent memory, indicates why it cannot follow the instruction that you have just given it:

PRINT ERROR REPORT



Instead of putting x on the screen, it has been hunting in vain for something in its memory, a variable.

Using the SYMBOL SHIFT key, now type in:

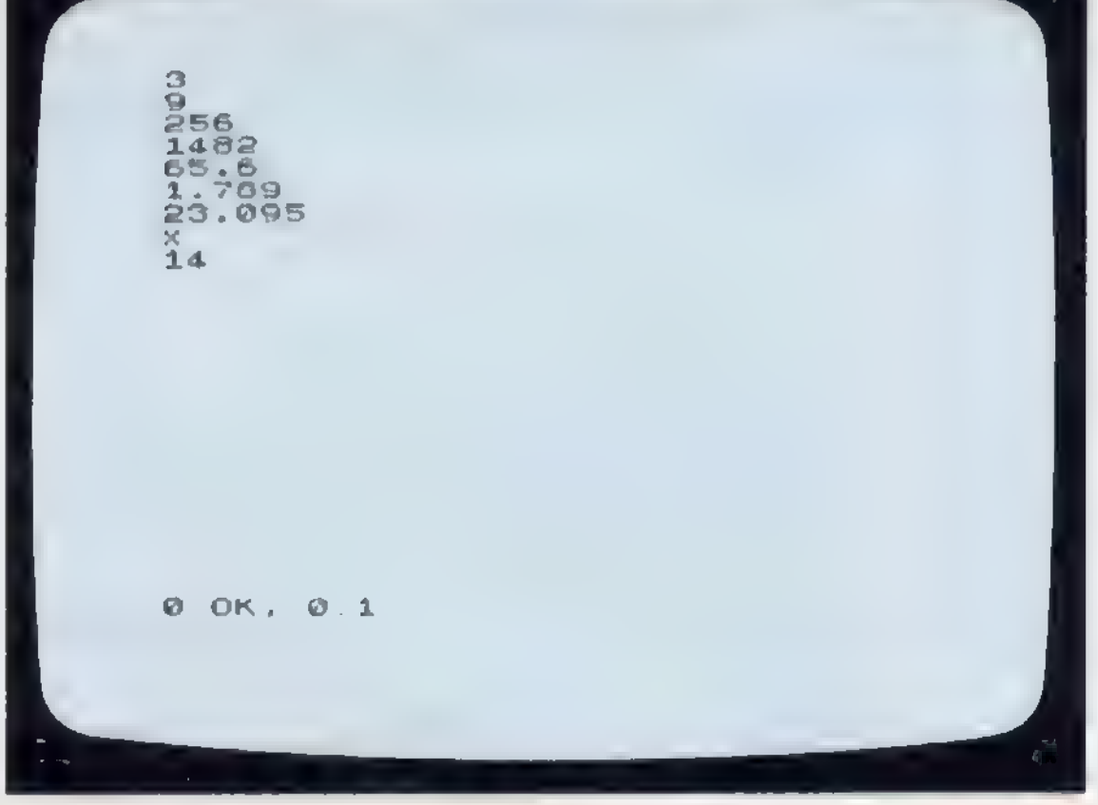
PRINT "x"

When you press ENTER, the computer makes the correct response – it PRINTs x at the next line.

You have just discovered that to the computer, x on its own and "x" mean two completely different things. The computer treats any letter on its own as a variable. A variable is simply a label identifying a number stored in the computer's memory. To make PRINT x comprehensible to the computer, give x a value (remember to press the ENTER key after each line):

LET x=14
PRINT x

USING **LET** AND **PRINT**



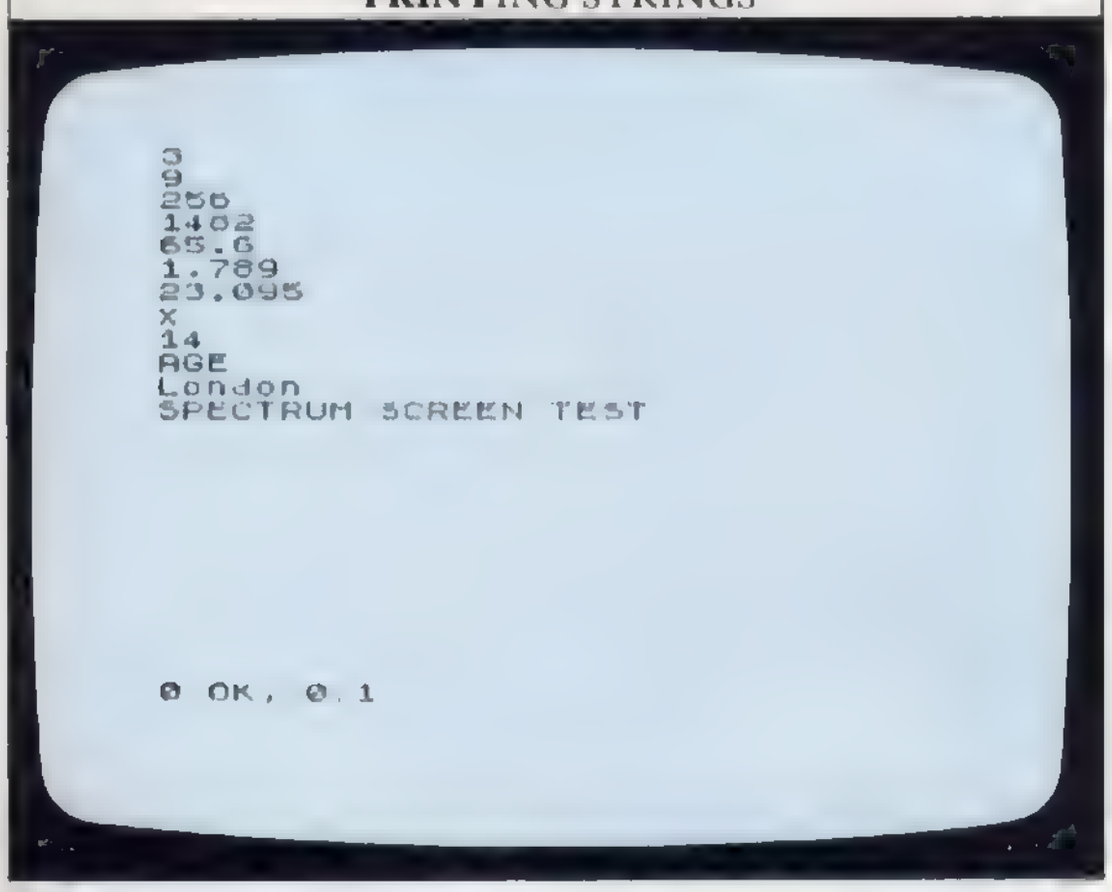
Now x is labelling something, the number 14. LET (on the L key) is a command which gives a label and a value to a slot in the memory. Every time you ask the Spectrum to PRINT x, it will display the last value keyed in. Because x is always a number, it is called a numeric variable.

How to use strings

So x is a numeric variable, but "x" is not; furthermore, even if you substituted a number for "x", it would not become a numeric variable, unless you removed the quotation marks. The computer displays everything inside quotation marks exactly as you type it. You can use any characters on the keyboard – letters, numbers, mathematical symbols and punctuation marks. Type these examples on your keyboard. Remember that you can use PRINT as many times as you like, as long as you press the ENTER key at the end of each command. You can pick capitals with CAPS SHIFT and CAPS LOCK:

```
PRINT "AGE"
PRINT "London"
PRINT "SPECTRUM SCREEN TEST"
```

PRINTING STRINGS



The characters between the quotation marks are called a string. In the same way as a number is stored in the computer and labelled by a numeric variable, a string is stored and labelled by a string variable. String variables are again always letters but unlike numeric variables they always end in a dollar sign. In the line:

```
LET A$="LONDON"
```

A$ is the string variable and LONDON is the string it labels. Having typed in the above line on the keyboard, clear the screen with CLS and then type:

```
PRINT A$
```

After you press the ENTER key, the computer will reveal the contents of the string variable typed in.

Positioning type with TAB and AT

Now try a different sort of PRINT command, this time with a sequence of strings:

```
PRINT "One", "Two", "Three", "Four", "Five",
"Six", "Seven", "Eight"
```

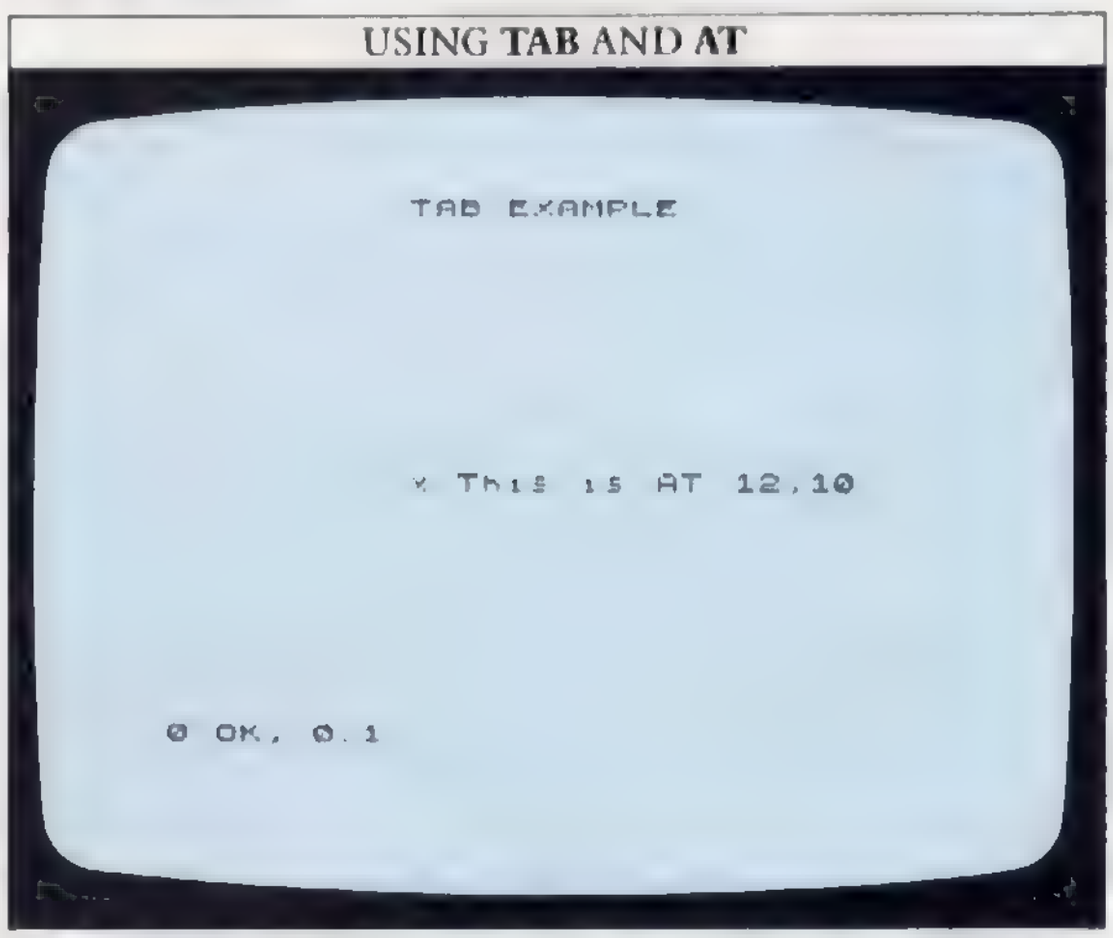
You will find that after you press ENTER the words are PRINTed in two columns. In fact, the screen is divided into two invisible fields, each 16 characters wide. "One" is PRINTed in the first field, "Two" in the next field, "Three" back in the first field, and so on.

However, two further commands, TAB (on the P key) and AT (on the I key), allow you to PRINT characters or strings at any position on the screen. Clear the screen with CLS and then type these two lines (you will need to type out everything between the quotation marks in full using letters):

```
PRINT  TAB 10;"TAB EXAMPLE"
PRINT  AT 12,10;"x This is AT 12,10"
```

If you press ENTER after each line, your screen should look like this:

USING **TAB** AND **AT**



This shows how TAB and AT position the text. TAB is used like the tab setting on a typewriter to PRINT at any position on a line. The number that follows it is the character position. There are 32 positions on a line – the TAB positions for them are numbered from 0 to 31, working from the left.

AT works like TAB, but it allows you to specify a vertical position as well, so you can PRINT at any point on the screen. AT is always used with a pair of numbers separated by a comma. The first number is the line number, working downwards from the top of the screen. There are 22 lines on the screen, numbered from 0 to 21. The second number is the character position, which works just like the number used with TAB. Always remember the comma when typing an AT command, otherwise the computer will not understand it.

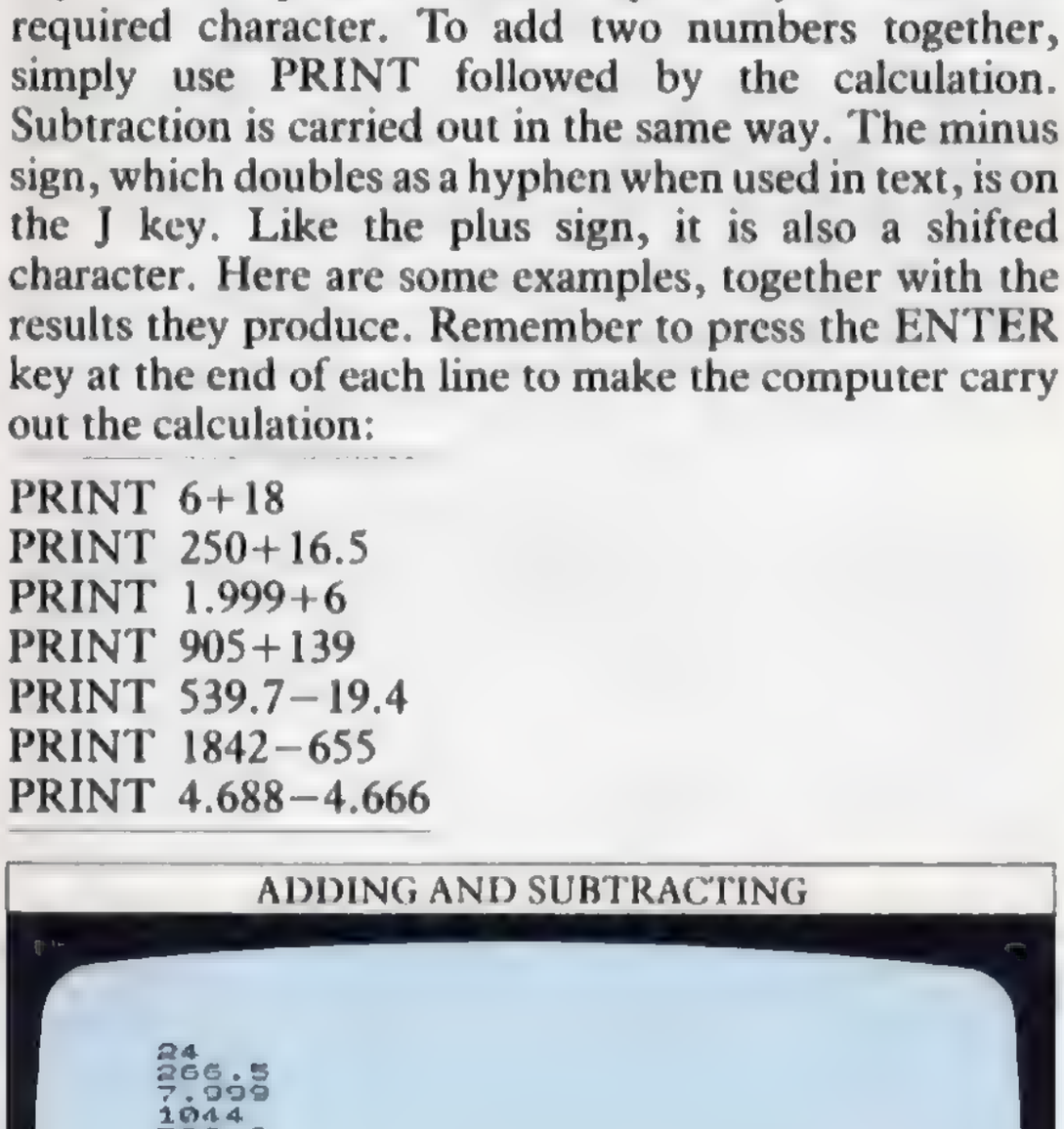
COMPUTER CALCULATIONS

The PRINT command is not limited just to displaying characters on the screen. You can also use it in conjunction with the four mathematical functions – addition, subtraction, multiplication and division – to perform calculations that you can follow on your television set.

Let's take addition first. The plus sign is on the K key, the second along from the ENTER key. Because it is the red symbol on the key-top, the SYMBOL SHIFT key must be pressed with the plus key to select the required character. To add two numbers together, simply use PRINT followed by the calculation. Subtraction is carried out in the same way. The minus sign, which doubles as a hyphen when used in text, is on the J key. Like the plus sign, it is also a shifted character. Here are some examples, together with the results they produce. Remember to press the ENTER key at the end of each line to make the computer carry out the calculation:

```
PRINT 6+18
PRINT 250+16.5
PRINT 1.999+6
PRINT 905+139
PRINT 539.7−19.4
PRINT 1842−655
PRINT 4.688−4.666
```

ADDING AND SUBTRACTING



Multiplication is carried out, not with the familiar ×, but with an asterisk, *. The asterisk is the red SYMBOL SHIFT character on the B key on the bottom row of the keyboard. Division uses the oblique stroke, /, a SYMBOL SHIFT character on the V key. In 24/8, for instance, the left-hand number is divided by the right-hand one. To key in the first of the following examples, press PRINT 3*6 and then ENTER:

```
PRINT 3*6
PRINT 14*9
PRINT 2.5*18
PRINT 5*78
PRINT 24/8
PRINT 366/3
PRINT 600/15
PRINT 100/0.01
```

MULTIPLYING AND DIVIDING



Exponents and square roots

In addition to these familiar maths functions, you can raise one number to the power of another, (called exponentiation). The keyboard cannot produce superscripts like the 3 in 2^3, so instead you have to use the up arrow (↑) symbol. 2^3 is calculated by PRINT 2 ↑ 3. Here are some examples of the up arrow in use:

EXPONENTS



The computer also allows you to find out the square root of a number. The command for this is SQR, the green function on the H key. SQR is used like this:

```
PRINT SQR 2
```

When you press ENTER after keying in this line, the computer will PRINT out the answer. However, if you try this command with a minus number, the computer will produce an error report to let you know that you have asked for a mathematical impossibility.

Getting the order right

You can carry out a number of different calculations using the same PRINT command. Try it with addition and subtraction first:

```
PRINT 2+6+3+7−8+3−4
PRINT 48−42+16−2
PRINT 122−19+32+2.5
PRINT 4.8+2.8+1.9
PRINT 1024+14−23
PRINT 15.5−12.5+7.6−3.8
PRINT 56−54+8+34−2+34+92−100+3
```

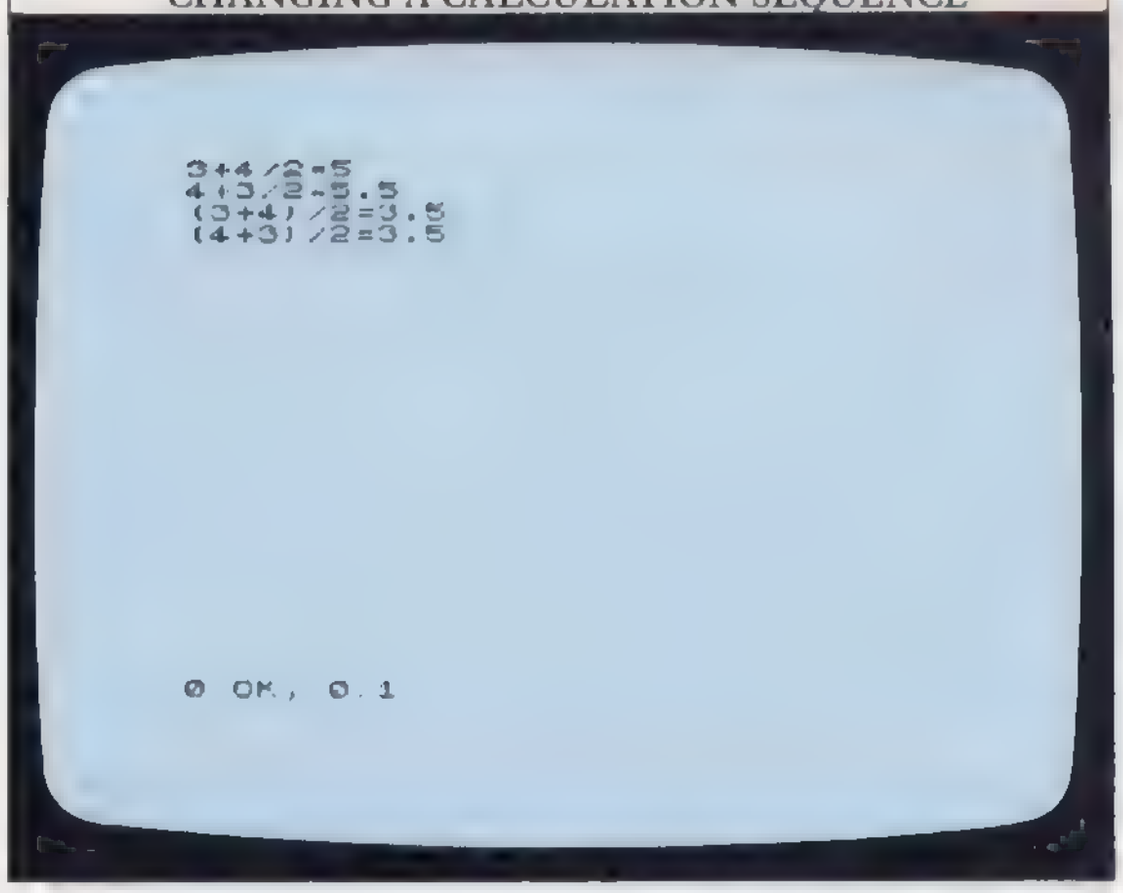
MULTIPLE CALCULATIONS



You can enter the figures for each calculation in any order at all, and the result will be the same. However, when you add multiplication and division to the chain of calculations, apparently odd things may happen. Look at the next list of examples, and try the calculations for yourself. Say you want to add two numbers together and divide the result by 2. The order in which the numbers are added should not make any difference to the result, but it appears to do so. Each of these lines PRINTs a calculation and result:

```
PRINT "3+4/2=";3+4/2
PRINT "4+3/2=";4+3/2
PRINT "(3+4)/2=";(3+4)/2
PRINT "(4+3)/2=";(4+3)/2
```

CHANGING A CALCULATION SEQUENCE



If 3+4 is exactly the same as 4+3, then why should the subsequent division by 2 make any difference? The reason is that the computer does not necessarily carry out calculations in the order in which you PRINT them on the screen. It performs exponentiation first, then multiplication and division, and finally addition and subtraction, always working from left to right. So in PRINT 3+4/2, 4 is divided by 2 before 3 is added. In PRINT 4+3/2, 3 is divided by 2 before 4 is added.

The problem you set the computer was to add two numbers together and then divide the result by 2. Neither of these examples does that. But you can change the order in which the computer performs calculations by enclosing parts of the calculation inside a pair of round brackets, as in the last two examples on the screen. Here, the addition within the brackets is carried out first and then the result is divided by 2.

Knowing your limitations

There are limits to the numbers that the Spectrum can handle and these limits take two forms – size and accuracy. The size limitation is unlikely to inconvenience you. Positive numbers can have any value from 4×10^{-39} (4 divided by 1 followed by 39 noughts) to about 10^{38} (1 followed by 38 noughts). The Spectrum stores these numbers to an accuracy of 9 or 10 figures. The computer memorizes just the first nine digits – the rest it sets at zero.

You may come across another of the computer's quirks when dealing with very big numbers. The Spectrum does not display them in the way in which you type them on the keyboard. For example, PRINT 2000000000000 produces 2E12 on the screen (the E stands for exponent). This is simply a shorthand way of displaying 2×10^{12} or 2 followed by 12 zeros, the number you keyed in. Try entering PRINT 10, PRINT 100, PRINT 1000 and so on and see how the computer deals with increasingly big numbers.

WRITING YOUR FIRST PROGRAM

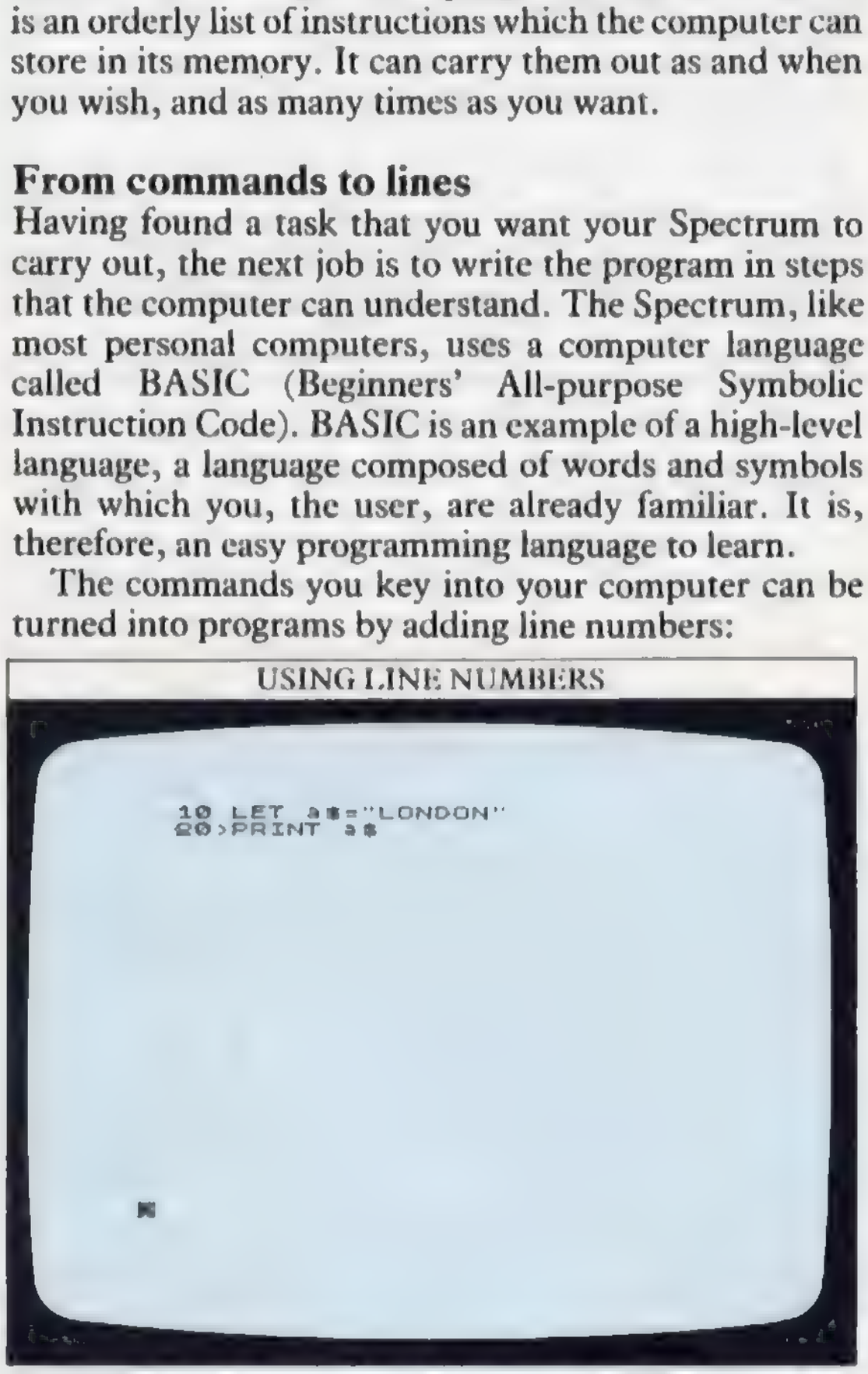
So far you have given your Spectrum commands to which it has responded immediately. These commands have been very simple – in many cases it would have been quicker not to have used the computer at all. However, commands on their own are not computer programs. The computer reads each command, carries it out and then forgets it. A program on the other hand is an orderly list of instructions which the computer can store in its memory. It can carry them out as and when you wish, and as many times as you want.

From commands to lines

Having found a task that you want your Spectrum to carry out, the next job is to write the program in steps that the computer can understand. The Spectrum, like most personal computers, uses a computer language called BASIC (Beginners' All-purpose Symbolic Instruction Code). BASIC is an example of a high-level language, a language composed of words and symbols with which you, the user, are already familiar. It is, therefore, an easy programming language to learn.

The commands you key into your computer can be turned into programs by adding line numbers:

USING LINE NUMBERS

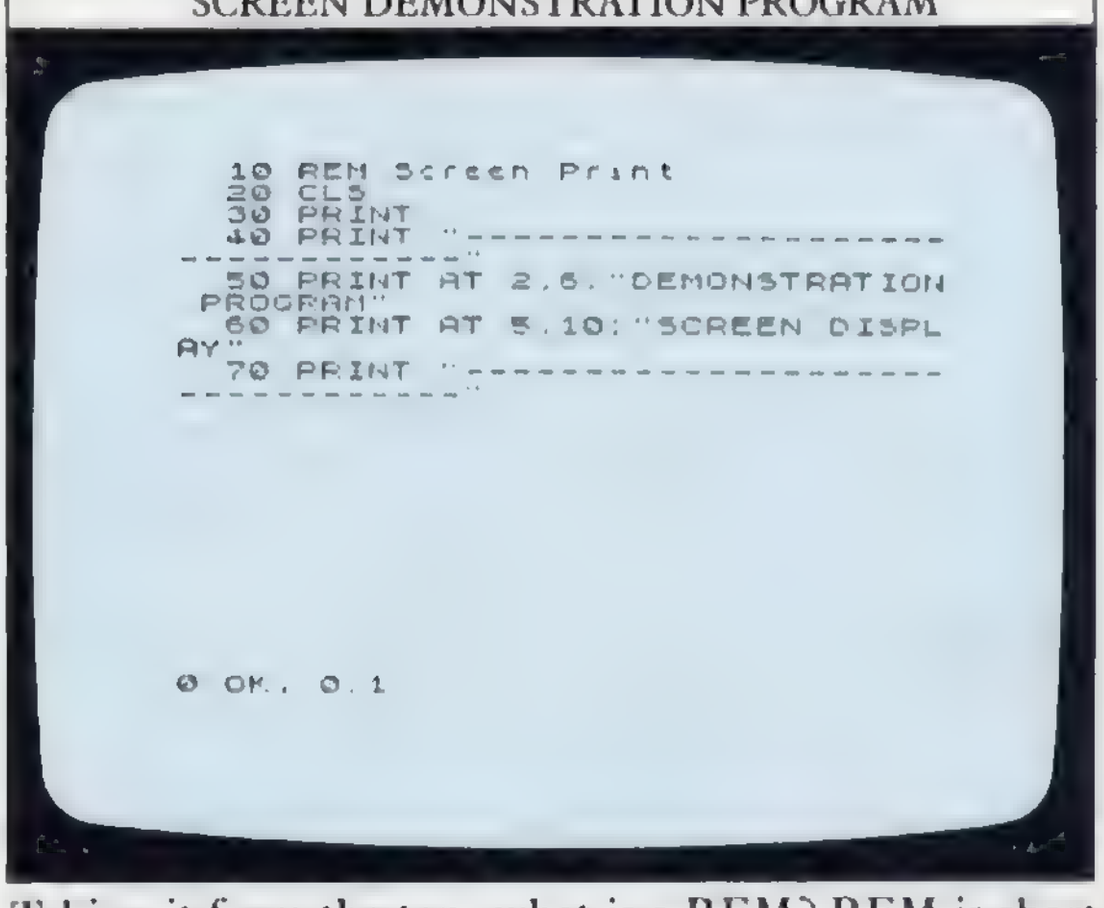


As you key the program in, you will notice that the commands are not now carried out as soon as you press the ENTER key, but instead remain displayed on the screen. Now that the program is safely stored in the computer's memory, it only remains to run it and see what it does. Do that by pressing RUN then ENTER.

You may be wondering why the lines are numbered 10, 20 and not 1, 2. When you are writing and testing programs, you will frequently want to add extra lines. If the existing lines are numbered 1, 2, 3, 4, and so on, there is nowhere to put the new lines in sequence. In the above program, there is room to add extra lines numbered 0–9 and 11–19, if necessary.

The program is still in memory, so to try the next one, switch off the power for a second or two to reset the computer and erase the old program. Then see what the following produces when you RUN it:

SCREEN DEMONSTRATION PROGRAM



Taking it from the top, what is a REM? REM is short for REMark. It's a useful device for titling or labelling parts of programs, so that you can find them again quickly. As your programming ability grows, you will find REM lines very useful for reminding you how a particular program works. Other people will also be able to follow your programs more easily if there are periodical REM statements to explain what you are doing. The computer doesn't do anything with a REM statement other than store it in memory.

CLS you have come across already. It's a quick way of taking all the old unwanted information off the screen. Using PRINT on its own (line 30) may at first seem a little crazy. PRINT tells the computer to send whatever follows it to the screen and move on to the beginning of the next line on the screen. If nothing follows PRINT, it just moves to the next line. Here, it is used to move the line of hypens down.

How to correct mistakes

It is easy to make mistakes on the keyboard which prevent the program from working. The Spectrum will not allow you to ENTER a line that contains a mistake in BASIC, but this doesn't stop you writing a program with correct lines that produces the wrong result. If you do notice a line that needs changing, there is no need to retype the program. The computer always uses the last version of any line, so to make a change, simply retype the line at the end of the program, press ENTER and the computer will insert it in the correct place.

Why punctuation is important

The next program uses the techniques demonstrated on pages 16–17. Switch off the power again for a second or two before keying it in:

CALCULATIONS PROGRAM

```
10 PRINT "3+10=",3+10
20 PRINT "6.25+4=",6.25+4
30 PRINT "179+9.5=";179+9.5

K
```

In each of the three calculations, the screen first shows what the calculation is (everything within the quotation marks is displayed exactly as it is written), and then the computer carries out the calculation itself and shows the result on the same line. The semi-colon is very important. It ensures that the result is shown on the same line as, and immediately following, the details of the calculation. Try the same program with a comma, a colon and nothing at all instead of the semi-colon. You'll quickly realize how important punctuation is in computer programming. Correct spacing is also vital if you want to produce a readable display when the computer PRINTs numbers and strings following each other. The following program, which again combines some calculations with PRINT, shows spaces within strings and how they appear when the program is RUN:

CONVERSIONS PROGRAM

```
10 PRINT "CONVERSIONS"
20 PRINT
30 PRINT
40 PRINT "1 foot=";12*2.54;" c
entimetres"
50 PRINT
60 PRINT "1 gallon=";8/1.76;"
litres"
70 PRINT
80 PRINT "10 Kilometres=";10*5
/8;" Miles"
90 PRINT
100 PRINT "1 day=";24*60*60;" s
econds"

K
```

CONVERSIONS DISPLAY

```
CONVERSIONS

1 foot=30.48 centimetres
1 gallon=4.5454545 litres
10 kilometres=6.25 miles
1 day=86400 seconds

0 OK, 100 1
```

How to write a flowchart

If a program is to RUN properly, it must carry out the correct operations in the right order. Drawing a flowchart is a useful way of outlining the steps involved in making the computer perform a task. This flowchart shows how to plan a program to add up all the numbers from 1 to 1000. Each shape is a separate operation, and the arrows connecting the shapes show the path that the program is to follow. NUMBER and TOTAL represent figures that can be entered in a program as numeric variables – n and t. This program contains two features which you will encounter later – a program "loop" and a program "decision point".

DRAWING A FLOWCHART

This chart shows all the steps needed to program a computer to add together all the numbers from 1 to 1000.

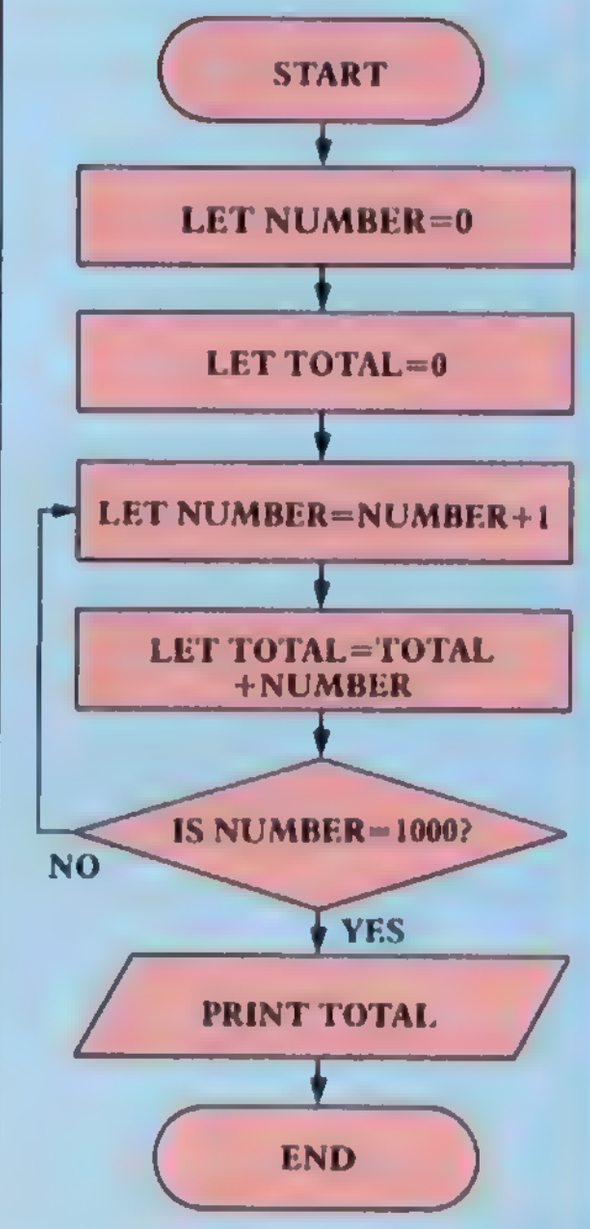
Key

Terminator Signals beginning and end of flowchart

Instruction Identifies each separate operation

Decision point Instructs computer to make a decision

Input/Output Instructs computer to take in or give out information



DISPLAYING YOUR PROGRAMS

As you start writing programs, you will often want to refer back to check on something or perhaps alter it in some way. In order to do that, you must be able to see the program on the screen again after it has been RUN. The Spectrum allows you look at anything you have stored in its memory. In this case, you want to look at the "program listing" – the program as you typed it in.

If you've just switched the computer on again after a short break, type in a program from a previous page. The BASIC word (or keyword) LIST will call up your program onto the screen again from the part of the memory where it is currently stored. Every time you press LIST the program will be displayed once more:

LISTING A PROGRAM



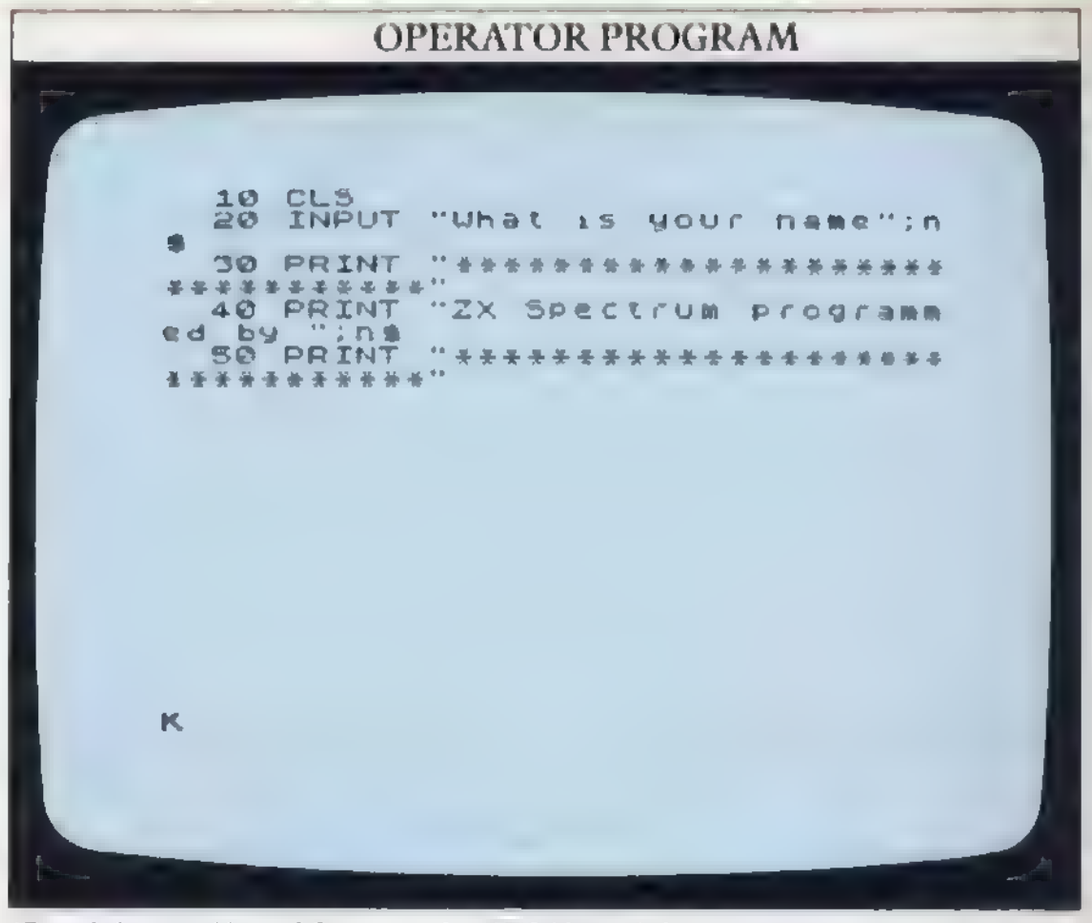
You haven't transferred the program from one part of the memory to the screen – it's still held in memory. You are now looking at a copy of it. If you want to make sure of that, use CLS again to clear the screen. If you now press LIST, the program will then reappear. You can do this as often as you like and the program will remain safely in memory unless you disconnect the computer.

Moving around a LIST

You will notice that when a program has been LISTed, it is displayed in an indented form, with the lines beginning two characters further to the right than they would if you had just keyed them in. Most of the programs in this book are shown in their LISTed form.

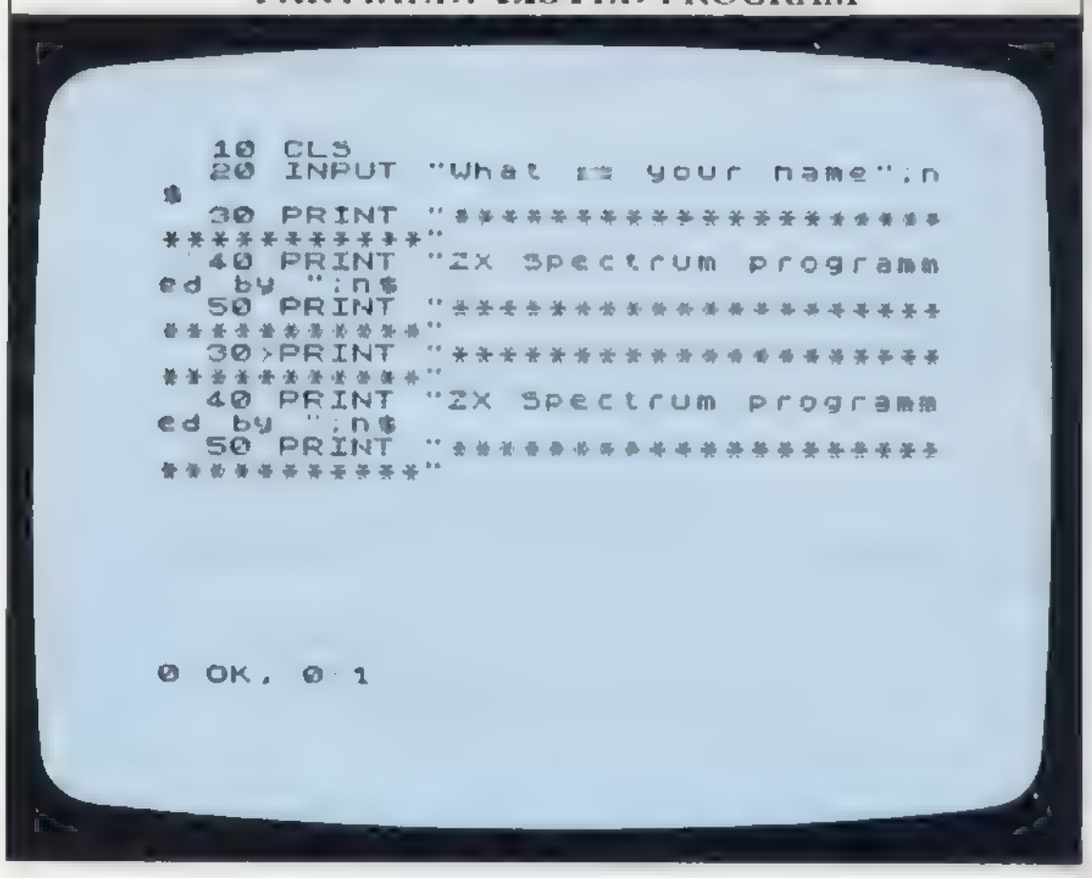
LIST is very useful for developing a program. All you have to do to see your program after RUNning is to type in LIST followed by the ENTER key. But LIST's capabilities do not end with producing a whole program for you to examine. If you key in the next program, you will be able to see how to use LIST more selectively. The program also demonstrates a technique which you will soon be using:

OPERATOR PROGRAM



(Incidentally, if you do RUN this program, press the ENTER key after you type in your name). Now if you want to display the whole program listing, type LIST. But you might only need to see a few lines from the end of a long program. Using the above program as an example, press LIST 30. Only lines 30 to the end of the program appear the second time:

PARTIALLY LISTED PROGRAM



How to enter a new program

Imagine you are starting a new program. Clear the screen and type in the first line:

```
10 PRINT "SPACE PROBE PROGRAM"
```

and RUN it. Something odd happens. The old program is still in memory. The computer PRINTs your new line 10, but then goes on to RUN the remainder of old program, because you haven't erased it:

COMBINED OLD AND NEW DISPLAYS



Up to now you have been switching off the power for a second or two before entering a new program, but there are better ways of getting rid of old programs. One of them is to use the BASIC keyword NEW. Press NEW and the ENTER key, then key in the new line 10. This time the old program will have gone for good.

RUNning a program segment

Programs may be RUN starting at any line number you like. Often you may have trouble getting part of a program to RUN properly. In a short program, it's just as convenient to RUN the whole program as it is to RUN only a part of it. But what if the troublesome part that you want to experiment with and RUN over and over again comes near the end of a long program? It soon becomes tiresome and time-wasting to have to watch the first five minutes or so of the program unfold before the suspect part comes up. Fortunately, you can jump part of a program by using RUN and a line number:

PARTIALLY **RUN** PROGRAM

```
  10 CLS
  20 INPUT "What is your name";n
$
  30 PRINT "**********************
**********"
  40 PRINT "ZX Spectrum programm
ed by ";n$
  50 PRINT "**********************
**********"
**********************************
ZX Spectrum programmed by David
**********************************


0 OK, 50 1
```

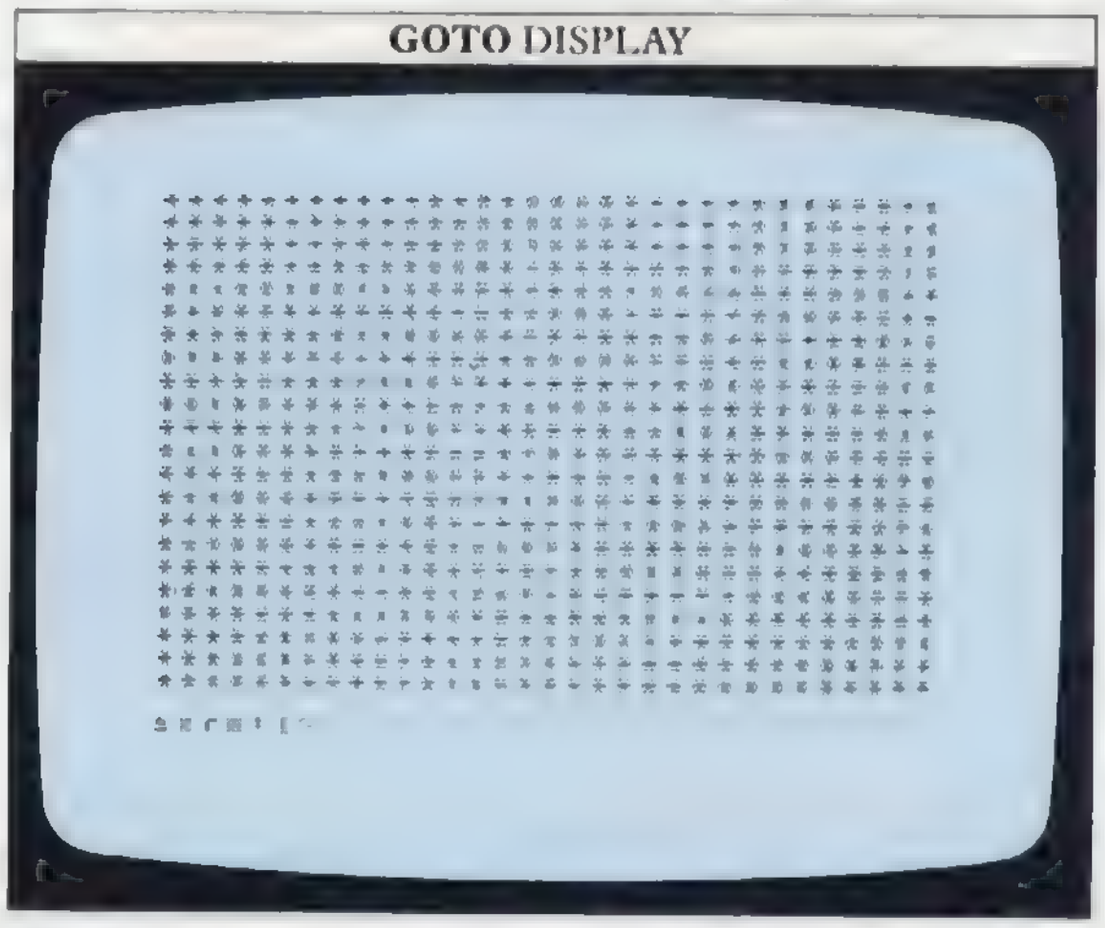
In this screen, the "operator" program from the opposite page has been LISTed and then followed by RUN 30 (produced by pressing RUN and 30 then the ENTER key). The result is that the computer goes straight to line 30, and then continues through the program as before. If you have already RUN the program, n$ will have a value – your name – and the computer will PRINT this out, although lines 10 and 20 will be ignored.

You can get exactly the same effect with the keyword GOTO. GOTO is one of the simplest and most useful commands in the BASIC language. Used without a program line in front of it, GOTO makes the computer go straight to a specified line and then RUN a program from that point. But when GOTO is actually part of a program, the results become very interesting. You can get an idea of what this command can do simply by keying in this simple program:

```
10 PRINT "*";
20 GOTO 10
```

Your screen should fill up with a display like this:

GOTO DISPLAY



If you are puzzled about why this has happened, don't worry. We will be returning to GOTO soon after you have mastered a few more BASIC keywords. However, you will find that it seems difficult to get rid of this display. The clue here is the word "scroll?" that appears at the bottom of the screen. The computer is asking you if you want the characters on the screen to continue to "scroll" upwards to fill the screen again because the program has not finished RUNning. In fact it has no end at all. To clear the screen press the N key. A message will then appear:

```
D BREAK – CONT repeats 10:1
```

Pressing any other key just makes the scrolling continue. Once this message has appeared, you can then LIST or erase the program with NEW.

CORRECTING MISTAKES

Computer programming is one pastime in which mistakes are unavoidable. Programs very rarely work satisfactorily first time, and the longer they are, the more difficult it is to get them right. It's important to realize that making mistakes and correcting them is often an interesting part of program development. So, don't ignore, hide or gloss over your mistakes – they are an invaluable aid to learning how to get things right.

For instance, in a computer program you cannot alter punctuation without completely changing the sense of what you have written. As you saw on page 19, it will have drastic results. To the computer, punctuation means something very precise, and if you get it wrong, a program may not work.

You can change a line in a program in two ways. First, as you have seen, you can simply retype the line. The new version automatically replaces the old one in the computer's memory. However, if there is very little wrong with a line, especially if it is a long line, it's a waste of time to completely retype it. The alternative way of making a change in this case is to use the cursor keys to edit on screen.

Editing on the screen

Editing involves using the four keys with arrows printed above them and the key with EDIT printed above it, all on the top row of the keyboard. Here is a program that needs editing:

PROGRAM BEFORE EDITING



To correct line 40 in the program to read:

40 PRINT "CHARLES DE GAULLE, PARIS"

you could retype the line. But try using the screen editor instead. First, type in the program and RUN it. Now LIST the program on the screen. Press the CAPS SHIFT key together with the key on the top row of the keyboard with a downward-pointing arrow (not the exponent key featured on page 16). The > symbol at the beginning of the last program line moves to line 10. Use the downward arrow to move it to the incorrect line. Now press CAPS SHIFT and EDIT (top row). The line marked by the > appears at the bottom of the screen:

PROGRAM DURING EDITING



You can now use the left- and right-pointing arrow keys on the top row to move the cursor to the end of the section to be deleted, after the word ROME. Now press CAPS SHIFT and DELETE (top row) repeatedly until ROME disappears. Then simply type in the characters that are to replace ROME. Finally press ENTER. It sounds complicated, but once you've tried it, it soon becomes second nature.

You will frequently want to add lines to a program after you have written the first draft. Perhaps you forgot to put CLS at the beginning to start the program off on a clear screen. You do not have to edit any line numbers to do this. In the above program, for example, you can enter the new first line by typing:

5 CLS

As the computer executes BASIC instructions in line number order, it doesn't matter that this line was added last – it will be carried out first.

First steps in bug-hunting

Mistakes in programs are called bugs, and the business of getting rid of them, debugging. As you have probably discovered, the Spectrum helps a great deal in debugging programs by examining what you type in for errors in spelling and grammar or syntax. If it finds any, it alerts you in two ways. First, if it comes across something that doesn't make sense in a line as you are typing it in, it will not let you ENTER the line. Pressing

ENTER will have no effect, and second, a question mark will flash next to the suspect part of the line, giving you a chance to correct it.

However, even if every single line of your program makes sense to the computer, the program may not RUN properly. You may have inadvertently told the computer to do something impossible – to add two numbers, A and B, when you haven't given A or B values, for instance. It responds to this by displaying an error report on the screen. You came across one on page 14 – "2 Variable not found, 0:1". In fact, the Spectrum can display over 25 different reports.

Each report begins with a number or letter (0–9 or A–R). The report itself is followed by something like 30:1. This means that the program has stopped at line 30. The "1" indicates that the error is in the first statement on the line. (As you'll see later, it is possible to write more than one instruction on a single program line). Here are some slightly more advanced programs which will not work. Try checking the error reports they produce on the table that follows them:

"BUGGED" PROGRAMS

```
10 FOR x=60 TO 100
20 PRINT 2↑x,3↑x
30 NEXT x

0 OK, 0:1
```

```
10 FOR f=1 TO 200 STEP 10
20 PLOT 150,f
30 DRAW 50,0
40 NEXT f

0 OK, 0:1
```

"BUGGED" PROGRAM

```
10 INPUT "Enter number ";a
20 FOR f=1 TO 12
30 PRINT TAB 9;f;"*";a;"=";f*a
40 NEXT g

0 OK, 0:1
```

ERROR TABLE

These are some error reports that you may encounter when writing your first programs.

Code	Report	Occurrence
0	OK	The "correct" report. The computer has found no errors in your program.
2	Variable not found	You have programmed the computer to do something with a variable without first defining it. This may occur if you miss out quotation marks before and after a string (see page 14).
4	Out of memory	All the available memory has been used up. With the 48K Spectrum this is unlikely to happen unless you try to combine a number of long programs with consecutive line numbers.
5	Out of screen	Your program has tried to INPUT more than a screenful of lines, or has tried to PRINT below the bottom line (see pages 15, 24–5).
6	Number too big	A calculation has produced a number bigger than the computer's limit of 10^{38} (see page 17).
A	Invalid argument	Your program has followed a function with an "argument" (the number on which the function is to operate) which cannot be used – for example SQR followed by a minus number (see page 17).
B	Integer out of range	Your program has produced a number that is larger than the limit that the computer can use for a certain operation. This often occurs with graphics (see page 28).
D	BREAK-CONT repeats	Appears when N, SPACE or STOP is pressed in response to "scroll?" (see page 21). This report also appears if BREAK is pressed while the computer is doing something other than carrying out a program – for example, LOADing a cassette (see page 60).
G	No room for line	The computer's memory has been filled, and there is no room for the line that you have just tried to ENTER.
L	BREAK into program	Appears when the BREAK key is pressed while a program is RUNning. The line and statement numbers which follow the report show the last line to be carried out. Pressing CONTINUE will make the computer carry on from that point.

COMPUTER CONVERSATIONS

In all the programs you have written so far, you have given the computer a set of instructions and left it to carry them out. Each program has had just one outcome, which was exactly the same every time the program was RUN. But few real programs are like this; in a games program for example, the players feed the computer with new instructions every time the game RUNs. The computer takes in these instructions during the course of the game, changing the display in response to the input of information.

Indeed, it is difficult to write a program of any complexity without being able to interrupt the program while it is RUNning to feed in new information.

The BASIC word INPUT is intended to deal with this situation. It lets you carry on a conversation of sorts with the computer – you "talk" to it through the keyboard and it "talks" to you through the screen.

The INPUT command makes the computer remember information typed in on the keyboard, and gives it a name – a numeric variable if the information is a number, or a string variable if the information is in string form. The information is then used later in a program. Here is an example of INPUT at work:

USING **INPUT**

```
 10 CLS
 20 PRINT "What is your name?"
 30 INPUT n$
 40 PRINT "**********************
**********"
 50 PRINT "ZX Spectrum programm
ed by ";n$
 60 PRINT "**********************
**********"




0 OK, 0 1
```

Questions from your computer

The program instructs the computer to display the question "What is your name?". Line 30 then stops the program, leaving the question PRINTed on the screen. The computer is waiting for new information from you. There's no need to hurry – there isn't a time limit. The computer will wait forever or until you type in the information it needs. Type in your name and press the ENTER key. The program continues.

The INPUT line of the program takes your name and labels it with the string variable n$. The dollar sign shows that the computer has been programmed to expect a string. This program is similar to the one used on page 20 as an example of LIST. You can see from that earlier program that INPUT can also PRINT:

COMBINING **INPUT** AND **PRINT**

```
 10 CLS
 20 INPUT "What is your name";n
$
 30 PRINT "**********************
**********"
 40 PRINT "ZX Spectrum programm
ed by ";n$
 50 PRINT "**********************
**********"




K
```

Programming multiple INPUTs

Many programs use INPUT a number of times to gather different items of information. It is quite easy to do this. All you have to remember is that you will need a separate variable for each INPUT. Once you have given the computer the information that each variable will label, it can then use the variables in a program.

In the previous program, n$ was used to label a string – in that case it was a name. But a string doesn't have to be just letters, it can be numbers. If you label a number as a string, the computer will deal with it as a string. Here is a program that does this:

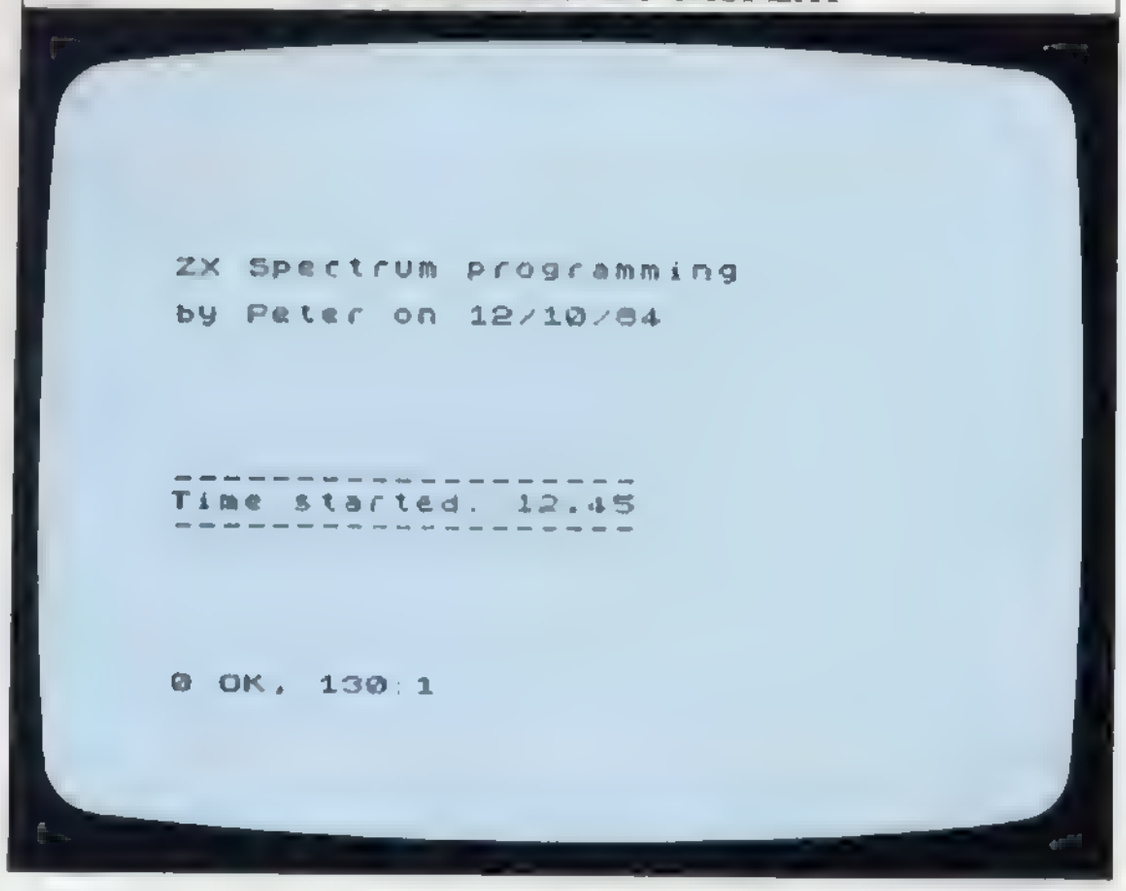
MULTIPLE **INPUT** PROGRAM

```
 10 CLS
 20 PRINT AT 2,9;"Enter name"
 30 INPUT n$
 40 PRINT AT 5,1;"Enter today's
date 00/00/00"
 50 INPUT d$
 60 PRINT AT 8,6;"Enter time 00
:00"
 70 INPUT t$
 80 CLS
 90 PRINT AT 5,0;"ZX Spectrum p
rogramming"
 100 PRINT AT 7,0;"by ";n$;" on
";d$
 110 PRINT AT 14,0;"------------
--------"
 120 PRINT AT 15,0;"Time started
: ";t$
 130 PRINT AT 16,0;"------------
--------"


0 OK, 0 1
```

In line 100, the computer PRINTs d$, which is the date. If the variable had been just d, the computer would have taken the date's oblique lines to mean "divide by". As a string, d$ is left unaltered:

MULTIPLE INPUT DISPLAY



Using INPUT with numbers

Because you can use INPUT to gather numbers as a program is RUN, this command has many practical applications. Consider, for example, the problem of converting lengths, sizes or weights from one unit of measurement into another. The conversion is always the same – 2.54 centimetres to the inch, 2.2 lbs to the kilogram, 1.76 pints to the litre, and so on – but the numbers in each new calculation are different. Here is a simple conversion program for you to try out:

INPUT CONVERSION PROGRAM

```
  10 CLS
  20 PRINT AT 5,0;"Conversion pr
ogram"
  30 PRINT AT 10,0;"How many pin
ts?"
  40 INPUT p
  50 PRINT AT 15,0;p;" pints=",p
/1.76;" litres"
```

```
0 OK, 0 1
```

The program asks you how many pints you want to convert to litres, waits for your response, does the calculation and then displays the result on the screen. Because the INPUT line is expecting a number in response to the question it asks, the variable it produces is a numeric one, p. This labels the number you key in for use later on in the program.

Next is a program that asks you for two pieces of information, and then uses them with the command AT to fix a point on the screen:

USING INPUT WITH AT

```
  10 CLS
  20 PRINT AT 5,5;"Mapping the s
creen"
  30 PRINT AT 10,0;"Give me a ro
w number (0-21)"
  40 INPUT r
  50 PRINT AT 15,0;"Give me a co
lumn number (0-31)"
  60 INPUT c
  70 CLS
  80 PRINT AT r,c;"X"
```

```
0 OK, 0 1
```

If you RUN this program, you will find that it asks you for a row and column number and then PRINTs an X at the position specified by your co-ordinates. The letters r and c are just labels, numeric variables waiting to be given values. It is these values that you key in when the program is RUN.

You can shorten this program by making a single line with one INPUT statement collect both these figures. Type in each number followed by ENTER:

COLLECTING TWO VARIABLES WITH ONE INPUT

```
  10 CLS
  20 PRINT AT 5,5;"Mapping the s
creen"
  30 PRINT AT 10,0;"Give me a ro
w number (0-21) and      a column
 number (0-31)"
  40 INPUT r,c
  50 CLS
  60 PRINT AT r,c;"X"
```

```
0 OK, 0 1
```

All the spaces between "and" and "a column" may look rather strange to you. If you don't put them in, you'll find that the message PRINTed by line 30 is split awkwardly between two lines. The extra spaces bring the whole of "a column number (0–31)" down to the middle of the next line.

WRITING PROGRAM LOOPS

Computers are extremely good at doing lots of simple, repetitive jobs very quickly. But if it is to do anything involving repetition, the computer must have some means of carrying out the same program or part of a program repeatedly. There are several ways of writing these program "loops". On page 21 you came across a loop using GOTO. Here it is in a slightly more complex loop produced by the same method:

NEVER-ENDING LOOP PROGRAM

```
10 CLS
20 LET x=1
30 PRINT x,x↑2
40 LET x=x+1
50 GO TO 30

0 OK, 0:1
```

```
 1            1
 2            4
 3            9
 4           16
 5           25
 6           36
 7           49
 8           64
 9           81
10          100
11          121
12          144
13          169
14          196
15          225
16          256
17          289
18          324
19          361
20          400
21          441
22          484

scroll?
```

If you RUN this program, you will quickly see the disadvantage of using GOTO alone – the program is never-ending. When the screen has filled up with figures, "scroll?" appears at the bottom to show you that the program has not finished yet.

If you press most keys, the display will continue. But if you do let the loop continue scrolling, there is a way that you can later exit from the program. As you saw on page 21, pressing the N key will break the loop. Using BREAK will also do this, producing a line number:

BREAKING INTO A LOOP

```
31            961
32            1024
33            1089
34            1156
35            1225
36            1296
37            1369
38            1444
39            1521
40            1600
41            1681
42            1764
43            1849
44            1936
45            2025
46            2116
47            2209
48            2304
49            2401
50            2500
51            2601
52            2704

L BREAK into program, 30:1
```

Note that in this loop program, line 50 does not point back to the very beginning of the program at line 10. If it did, x would always be equal to 1, and the screen would clear each time the first line was PRINTed.

How to stop a loop

The solution to these endless programs is the FOR ... NEXT loop. This allows you to set limits on how many times the program is carried out. You can adapt the GOTO program to use FOR...NEXT instead:

FOR...NEXT LOOP PROGRAM

```
10 CLS
20 FOR x=1 TO 21
30 PRINT x,x↑2
40 NEXT x

0 OK, 0:1
```

The FOR ... NEXT loop both improves the program and shortens it by one line. Note that you don't have to include LET x= or add 1 to x on each loop of the program now, because FOR ... NEXT takes care of it automatically. It starts off by setting x equal to 1 and PRINTing x and x-squared. Line 40 asks for the next value of x and so the program re-starts from line 20, the

beginning of the FOR ... NEXT loop, and executes the intervening lines once more. This continues until x has a value of 21, the maximum set by line 20, and in this case, the program stops.

If necessary, the loop can be interrupted on each pass through to wait for new information. Try using INPUT in the middle of a FOR ... NEXT loop:

FOR...NEXT WITH INPUT

```
   10 FOR n=1 TO 5
   20 CLS
   30 PRINT AT 5,5;"Temperature c
onversion"
   40 PRINT AT 12,0;"Type in a Fa
hrenheit temperature"
   50 INPUT t
   60 PRINT AT 14,0;t;" Fahrenhei
t=";(t-32)*5/9;" Centigrade"
   70 PAUSE 200
   80 NEXT n


0 OK, 0:1
```

This program converts Fahrenheit temperatures into Centigrade. The FOR ... NEXT loop beginning at line 10 sets a limit of five calculations, after which you will have to RUN the program again. The INPUT statement at line 50 stops the program until you type in the Fahrenheit temperature you want to convert. Line 60 then does the calculation and PRINTs the result.

Slowing a loop down

One problem you may encounter when writing programs is that they often RUN too fast for you to be able to follow anything which is PRINTed on the screen. Line 70 in the temperature conversion program is used to deal with this problem. The command PAUSE stops the program temporarily so that the result of the conversion stays on screen long enough for you to read it before the next run through the loop begins and the screen is cleared. The length of this halt is set by the number following PAUSE. This number represents the length of the PAUSE in fiftieths of a second. Hence PAUSE 200 interrupts the program for 200/50ths or 4 seconds whereas PAUSE 0.5 is just 1/100th of a second.

How to round numbers off

The layout of the conversion display could be improved. It's fine as long as the result of the calculation is a whole number, but it rarely is. The more figures there are after the decimal point, the further "Centigrade" is pushed along the line until it splits and part of it ends up on the next line:

TEMPERATURE CONVERSION DISPLAY

```
Temperature conversion


Type in a Fahrenheit temperature
65 Fahrenheit=18.333333 Centigra
de
```

To get around this problem, try replacing (t−32)*5/9 by INT ((t−32)*5/9+0.5). INT, short for INTeger, turns a decimal number into a whole number. If the result is 18.333333, for instance, adding INT changes that to 18. The number is more sensible, and the display looks much better:

ROUNDED-OFF CONVERSION DISPLAY

```
Temperature conversion


Type in a Fahrenheit temperature
65 Fahrenheit=18 Centigrade
```

When you use INT, remember that it always rounds downwards to the next whole number. You may have wondered why the INT line has 0.5 before the last bracket. This is to ensure that INT always produces the nearest whole number. Adding 0.5 will achieve this. If you are confused by this, try keying in these two direct commands:

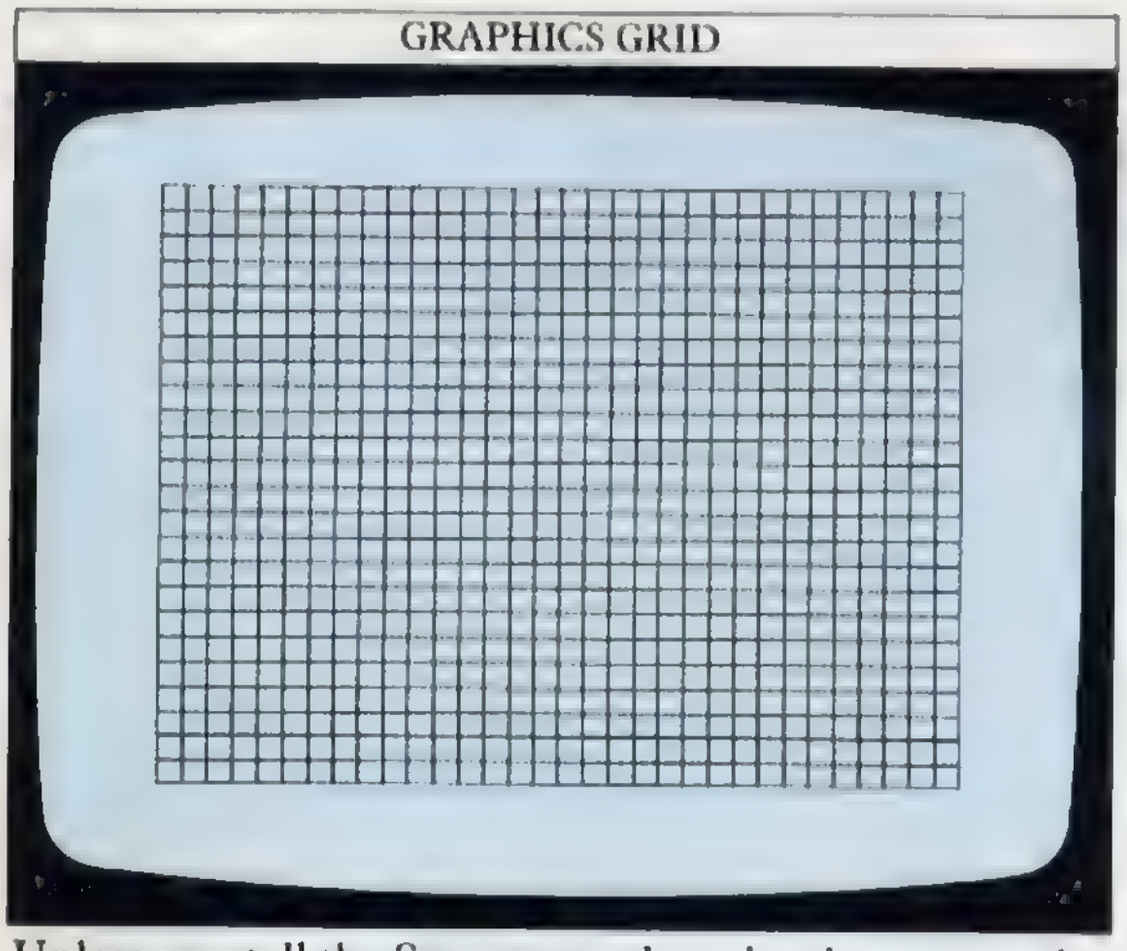
```
PRINT 3−1.1
PRINT INT (3−1.1)
```

The result of the first is 1.9, and the result of the second 1. But 1.9 is much nearer to 2 than 1. So, to compensate for this in the temperature conversion program, 0.5 is added before INT is used.

THE ELECTRONIC DRAWING-BOARD

The ZX Spectrum's BASIC includes several commands for drawing points and lines on the screen. If you want to draw a point or line, you must have some way of telling the computer where to start drawing. The screen is, therefore, divided up into a grid, made up of tiny dots. These dots are called picture elements (usually shortened to pixels) because the picture on the screen is made up from them. Each pixel is numbered – 0 to 255 across, and 0 to 175 up and down. The 0,0 position is at the bottom left corner of the screen. The co-ordinates of a point on the screen are always given as x,y, with x (the number of pixels across the screen from the left) first, followed by y (the number up from the bottom of the screen). On the screen below, each grid square is 8 pixels wide and 8 pixels high:

GRAPHICS GRID



Unless you tell the Spectrum otherwise, it assumes that you want to start drawing from 0,0 (also called the "origin") or from the last position visited. You can make a point appear on the screen by using the Spectrum's PLOT command. To make a black dot appear at the centre of the screen, type:

```
PLOT 128,88
```

You can check these co-ordinates on page 61.

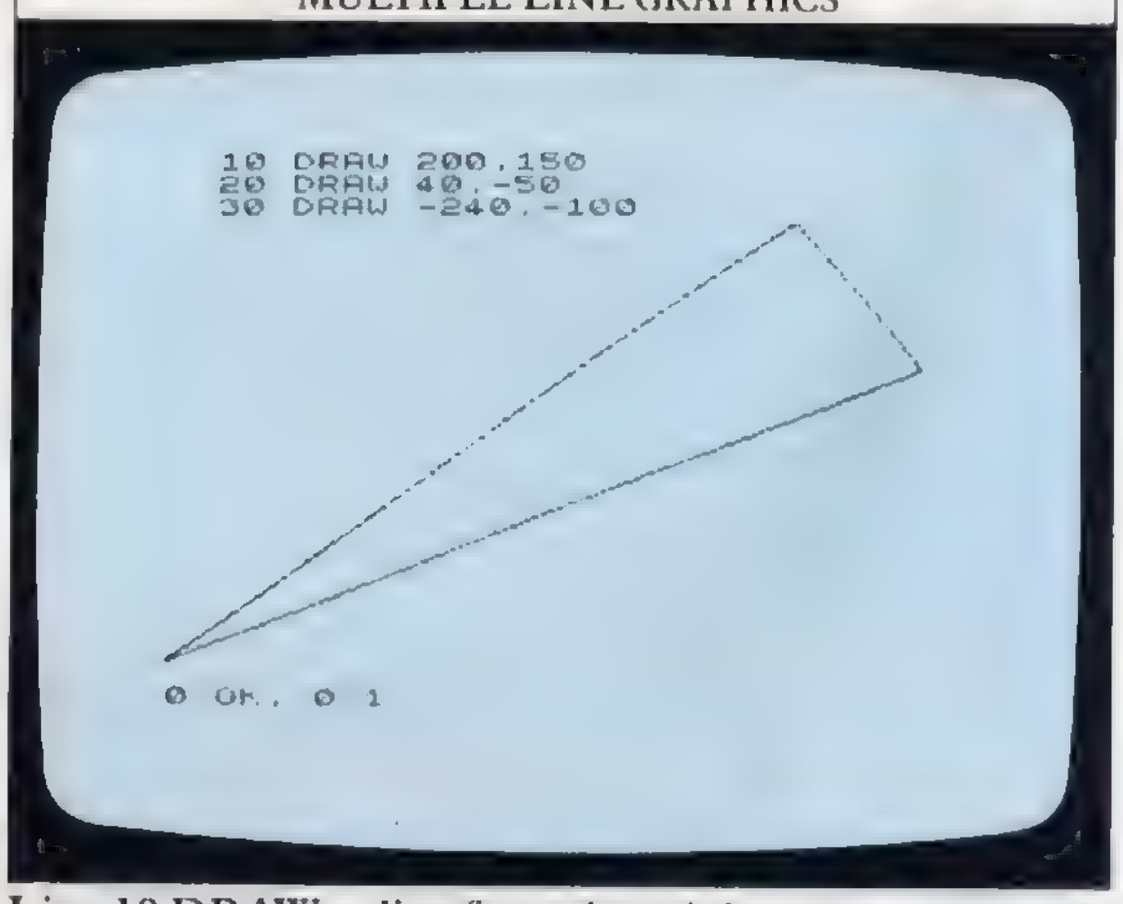
How to draw lines

Lines are drawn on the screen using the command DRAW. For instance:

```
DRAW 128,88
```

DRAWs a line from the bottom left corner of the screen, the origin, to a point in the middle of the screen. If you then instruct the computer to DRAW a second line, it will automatically start DRAWing from wherever the last line ended. You can use this to your advantage to produce simple, straight-line shapes very easily. Here is a LISTed program that produces three lines, together with the shape it DRAWs:

MULTIPLE LINE GRAPHICS

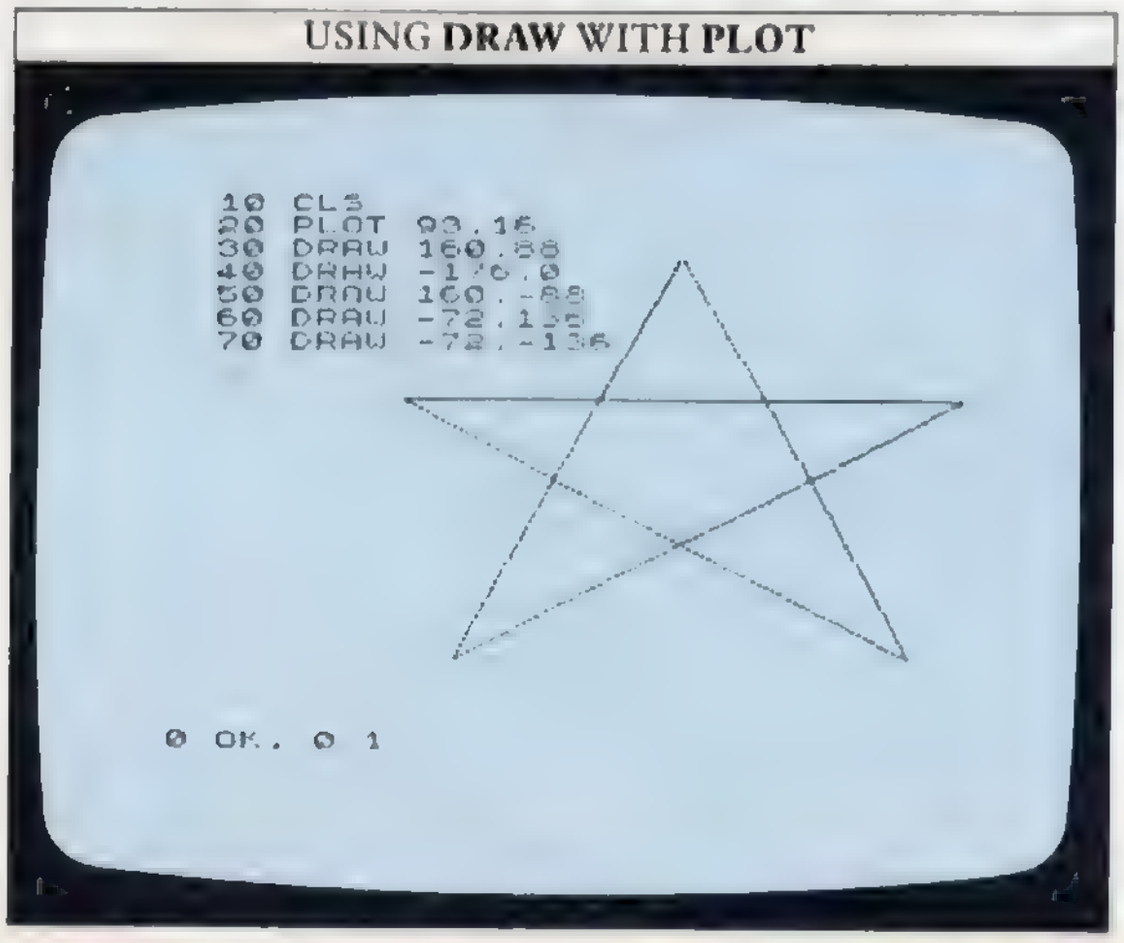


Line 10 DRAWs a line from the origin at 0,0 to 200,150. The next line is to be DRAWn from 200,150 to 240,100. The changes in the x and y co-ordinates are 40 and −50 respectively, so line 20 is therefore DRAW 40, −50. Finally, line 30 DRAWs the third side of the triangle from 240,100 back to the origin.

Picking a starting point

To DRAW a shape in the middle of the screen, you must tell the computer that you want to start DRAWing from somewhere other than the origin at 0,0. The PLOT command takes care of that. First, PLOT a point where you want to begin DRAWing. Now, the next line DRAWn will begin from that point:

USING **DRAW** WITH **PLOT**



How to fill in shapes

It is now a simple matter to fill in these line drawings to produce solid black figures:

SOLID RECTANGLES WITH **FOR...NEXT**

```
10 FOR x=20 TO 220
20 PLOT x,100
30 DRAW 0,-80
40 NEXT x
```

```
0 OK, 0:1
```

This repeatedly DRAWs lines from the top of the rectangle down to the bottom, gradually working from left to right, generating a solid black rectangle. A solid triangle is produced in a different way. The lines are all DRAWn from a single point, one apex of the triangle:

SOLID TRIANGLES WITH **FOR...NEXT**

```
10 FOR x=1 TO 110
20 PLOT 128,88
30 DRAW x-128,-88
40 NEXT x
60 FOR x=120 TO 255
70 PLOT 200,150
80 DRAW x-200,-150
90 NEXT x
```

```
0 OK, 0:1
```

Now let's try something a little more ambitious. In addition to points and lines, the Spectrum can generate a number of graphics characters that are permanently stored in its memory. You can see them printed on the tops of keys 1 to 8. The largest is a square the size of one character, while the others display halves, quarters and other fractions of squares. Using these characters is a simple way of generating graphics, but it produces only coarse images, and, since each character can occupy only character positions, only jerky movements are possible when you come on to animation.

Selecting them involves switching to the graphics cursor. For instance, to make a black square type:

```
PRINT "■"
```

To produce this, before pressing key 8, switch to the graphics cursor by pressing CAPS SHIFT and 9. Now, holding CAPS SHIFT down, press key 8 to produce the character. Before continuing, press CAPS SHIFT and 9 again to remove the graphics cursor. You can produce the inverse (negative) of any keyboard graphic by not using CAPS SHIFT again when pressing the symbol key. So, to summarize, the sequence of key presses needed to produce the black square is:

CAPS SHIFT and 9
CAPS SHIFT and 8
CAPS SHIFT and 9

DRAWing a simple landscape

The following program shows you how you can use these graphics symbols together with the PLOT and DRAW commands to produce a basic landscape:

USING KEYBOARD GRAPHICS

```
10 FOR c=15 TO 21
20 FOR r=1 TO 21
30 PRINT AT r,c;"■"
40 NEXT r
50 NEXT c
60 FOR x=-32 TO 32
70 PLOT 40,88
80 DRAW x,10
90 NEXT x
100 FOR x=-90 TO 64
110 PLOT 128,88
120 DRAW x,-32
130 NEXT x
140 FOR x=-50 TO 50
150 PLOT 170,24
160 DRAW x,-16
170 NEXT x
180 PRINT AT 5,25;"■"
190 PRINT AT 4,24;"■"
200 PRINT AT 4,25;"■"
210 PRINT AT 6,24;"■"
220 PRINT AT 6,26;"■"
```

```
0 OK, 0:1
```



```
0 OK, 220:1
```

DESIGNING YOUR OWN CHARACTERS

You can make up almost any shape of graphics character by using PLOT and DRAW as described on pages 28–29. However, it is much more convenient if you store your own graphics characters in the computer's memory in the same way as it stores all of its own keyboard characters. You can then treat a graphic symbol – a rocket ship or a human figure – as a single character, instead of having to RUN a special program to DRAW and PLOT the symbol each time you need it. The Spectrum allows you to program the A to U keys with any graphics symbols of your choice.

How to use a character grid

Say you want to put a small rocket on the screen for a space game. The shape will be shown on the screen as a pattern of dots on an 8×8 grid, so draw one out on an 8×8 grid on a piece of paper, as on the diagram below, or use the grid on page 61.

Draw the shape by blacking in whole squares. When you have done this, add up the numerical values of the squares in each row. On the top row of the example, only one square, the one with 8 above it, is blacked in, so the total for the top row is 8. In the next row, the squares labelled 4, 8 and 16 are black, so the total for row 2 is 28, and so on. These totals can then be fed into the computer to reprogram one of its keys.

POKE is a command which allows you to put information directly into the computer's memory. USR"a" in the following program tells the computer that you want to recover your character by pressing the "a" key. (Using a capital A would work equally well). The number following (+1, +2, etc) identifies which row of the 8×8 grid the final number refers to. Eight lines of the program are necessary to install the new character, working from "a" through to "a"+7. Here is the rocket and the program that stores it:

SINGLE GRID CHARACTER

Individual square values

128	64	32	16	8	4	2	1	Row Totals	
				■				8	= 8
			■	■	■			16+8+4	= 28
		■		■		■		32+8+2	= 42
	■		■	■	■		■	64+16+8+4+1	= 93
			■	■	■			16+8+4	= 28
			■	■	■			16+8+4	= 28
		■	■	■	■	■		32+16+8+4+2	= 62
	■	■	■	■	■	■	■	64+32+16+8+4+2+1	=127

ROCKET PROGRAM

```
10 POKE USR "a",8
20 POKE USR "a"+1,28
30 POKE USR "a"+2,62
40 POKE USR "a"+3,93
50 POKE USR "a"+4,28
60 POKE USR "a"+5,28
70 POKE USR "a"+6,62
80 POKE USR "a"+7,127
```

```
0 OK, 0 1
```

Now you need a way of getting your character back out of memory and onto the screen. The PRINT statement is used. To PRINT the newly programmed character in the middle of the screen add this line to the above program, first changing to the graphics cursor, and then repeatedly pressing the A key:

```
90 PRINT AT 15,8;"AAAAAAAAAAAAA"
```

ROCKET DISPLAY

```
          ▲▲▲▲▲▲▲▲▲▲▲▲▲

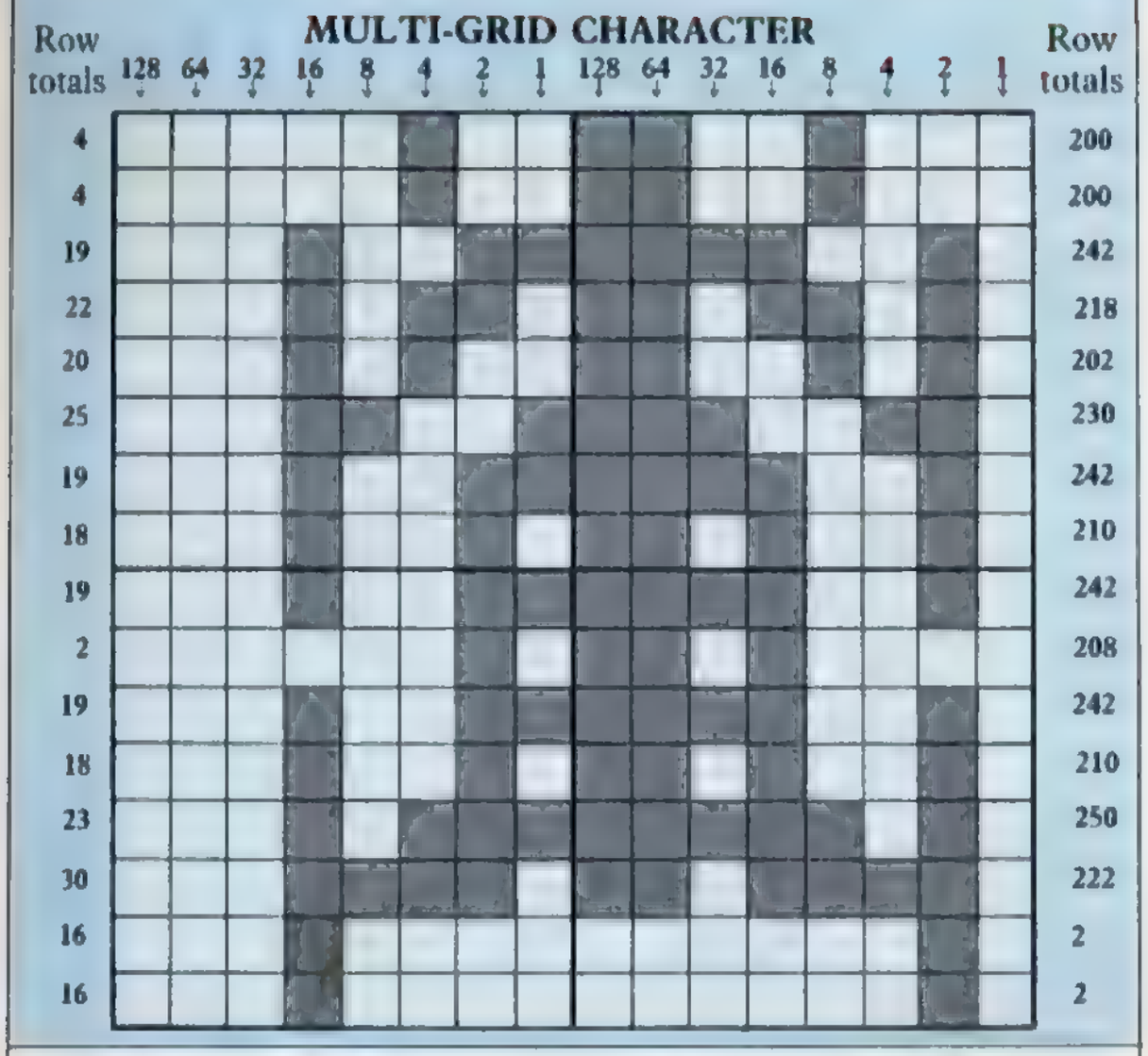

0 OK, 90:1
```

By switching to the graphics cursor, pressing the A key repeatedly, and then removing the graphics cursor again as described on page 29, a row of the character you have designed is entered in the program line. Then just RUN the program.

How to add characters together

The first thing you will notice is that the rockets are extremely small. Have a try at something larger:

MULTI-GRID CHARACTER



Although user-defined characters are based on an 8×8 grid, there's no reason why your character should not cover more than one grid.

Again, key in the totals representing each row in each 8×8 grid. Each grid must be labelled with a different keyboard letter to identify it. In the program that follows the new rocket, they are labelled with the letters a, s, d, and f.

Screen patterns with POKE USR

With user-defined graphics it is possible to use your own characters to make a pattern by PRINTing them all over the screen. In this program, try filling in the zeros in lines 10–80 to define a character (the grid on page 61 will help you with this). Instead of giving the character a fixed position such as in the above example, the program will PRINT the character AT r,c where r (the row number) and c (the column number) are constantly changing. Here is the program and a display made by replacing all the zeros in lines 10–80 with 195,231,126,36,36,126,231,195:

ADDING CHARACTERS TOGETHER

```
 10 POKE USR "a",4
 20 POKE USR "a"+1,4
 30 POKE USR "a"+2,19
 40 POKE USR "a"+3,22
 50 POKE USR "a"+4,20
 60 POKE USR "a"+5,25
 70 POKE USR "a"+6,19
 80 POKE USR "a"+7,18
 90 PRINT AT 11,16;"A"
100 POKE USR "s",200
110 POKE USR "s"+1,200
120 POKE USR "s"+2,242
130 POKE USR "s"+3,218
140 POKE USR "s"+4,202
150 POKE USR "s"+5,230
160 POKE USR "s"+6,242
170 POKE USR "s"+7,210
180 PRINT AT 11,17;"S"
190 POKE USR "d",19
200 POKE USR "d"+1,2
210 POKE USR "d"+2,19
220 POKE USR "d"+3,18

Scroll?
```

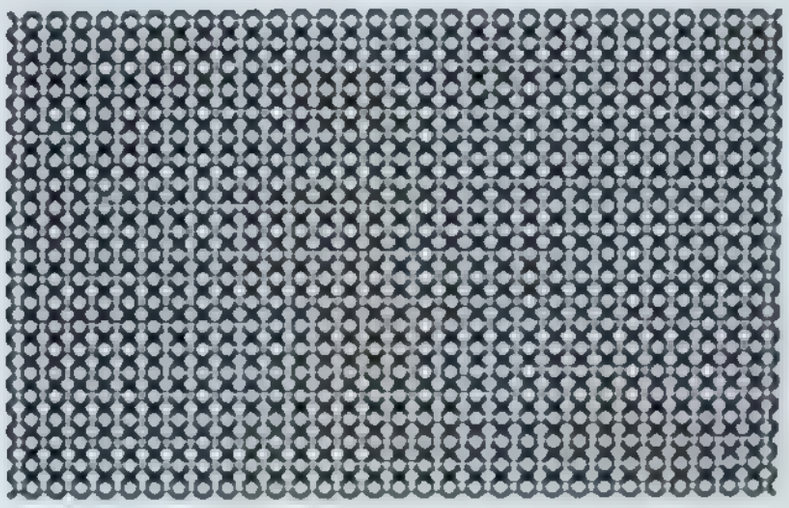
```
230 POKE USR "d"+4,23
240 POKE USR "d"+5,30
250 POKE USR "d"+6,18
260 POKE USR "d"+7,16
270 PRINT AT 12,16;"D"
280 POKE USR "f",24
290 POKE USR "f"+1,60
300 POKE USR "f"+2,24
310 POKE USR "f"+3,24
320 POKE USR "f"+4,24
330 POKE USR "f"+5,60
340 POKE USR "f"+6,24
350 POKE USR "f"+7,60
360 PRINT AT 12,17;"F"

0 OK, 0:1
```

PATTERN GENERATOR PROGRAM

```
 10 POKE USR "a",0
 20 POKE USR "a"+1,0
 30 POKE USR "a"+2,0
 40 POKE USR "a"+3,0
 50 POKE USR "a"+4,0
 60 POKE USR "a"+5,0
 70 POKE USR "a"+6,0
 80 POKE USR "a"+7,0
 90 FOR r=1 TO 21
100 FOR c=1 TO 31
110 PRINT AT r,c;"A"
120 NEXT c
130 NEXT r

0 OK, 0:1
```



```
0 OK, 130:1
```

ANIMATION

Once you have learned how to create a character and make it appear anywhere on the screen using PRINT AT, you can then attempt some simple animation by using the same method to make the character move. First, you need a program that produces a character:

STATIC CHARACTER

```
 10 POKE USR "q",36
 20 POKE USR "q"+1,126
 30 POKE USR "q"+2,219
 40 POKE USR "q"+3,126
 50 POKE USR "q"+4,36
 60 POKE USR "q"+5,36
 70 POKE USR "q"+6,255
 80 POKE USR "q"+7,255
 90 POKE USR "r",255
100 POKE USR "r"+1,90
110 POKE USR "r"+2,90
120 POKE USR "r"+3,90
130 POKE USR "r"+4,90
140 POKE USR "r"+5,90
150 POKE USR "r"+6,219
160 POKE USR "r"+7,219
170 PRINT AT 11,15;"Q"
180 PRINT AT 12,15;"R"


0 OK, 0:1
```

```
0 OK, 160:1
```

This program creates an "alien" with two graphics symbols by reprogramming the Q and R keys. Together, the symbols make up an alien in the middle of the screen. Line 170 tells the computer to PRINT the top half of the alien AT 11,15, while line 180 uses AT to put the bottom half in the right place.

Now you can try making it move from one side of the screen to the other. You will be relieved to know that lines 10 to 160 inclusive remain exactly the same, so don't press NEW before making the following changes or you will have to type in the whole of the program again. There's no need to delete lines 170 or 180, because the new lines will replace them.

These lines and changes will start the alien moving (the graphics symbols will automatically replace Q and R if you have RUN the original program):

ANIMATION LINES

```
170 LET c=11
190 FOR c=30 TO 30
1900 PRINT AT r,c;"Q"
2000 PRINT AT r+1,c;"R"
210 NEXT c


0 OK, 0:1
```

How to remove after-images and control speed

When you RUN the modified program, the first thing you will notice is that it doesn't do exactly what you want it to. The Spectrum moves the alien from left to right, but as it moves the alien, it doesn't remove the old images. The result is a long line of characters:

DISPLAY WITH AFTER-IMAGES

```
0 OK, 210 1
```

The problem is that you haven't told the computer to remove the old unwanted images. Doing that is quite easy – just PRINT a blank space after each of the characters with these lines:

```
205 PRINT AT r,c-1;" "
206 PRINT AT r+1,c-1;" "
```

The alien now moves from left to right without leaving a trail. But it's very fast. How can you scale down the speed? Put in a time delay:

```
207 PAUSE 2
```

Now the alien takes longer to reach the right side of the screen. You can adjust its speed by altering the PAUSE. This screen shows an impression of the movement (the after-images will not actually appear on your screen):

ANIMATION WITH DELETIONS



You will find that the slower the speed, the more clearly the alien appears on the screen. Flickering occurs when a symbol is moving quickly, or if it is made up of a number of characters. With a two-character symbol like this, there is a slight time delay between the computer PRINTing the first and the second character. If you use more than two characters to make up a symbol, the flickering as the symbol moves will be more noticeable.

Movement up and down the screen

You can move something down the screen from top to bottom, or vice versa, equally easily. Instead of varying the horizontal position (shown by the changing variable c in the alien program), you change the vertical position by the same method.

The next program stores a symbol representing a small rocket under the A key, and then makes it fall down the screen. The same sort of FOR ... NEXT loop as was used in the alien animation is used here at line 140. (FOR...NEXT and other program loops are explained on pages 26–27.) The r (row) variable increases so that the rocket is PRINTed at progressively lower positions.

To make sure that the rocket does not leave a trail, the program PRINTs a blank space behind it. Because this program uses a symbol only one character wide, only one blank space is needed. To give the rocket something to hit, lines 90 to 120 PRINT a simple landscape with a horizon, just like on page 29. Line 190, which is only carried out after the FOR ... NEXT loop has been completed, PRINTs five graphics characters (ones found on the number 6 key). This line produces the impression of the rocket's impact:

ARROWHEAD FALL PROGRAM



```
10 POKE USR "a",255
20 POKE USR "a"+1,255
30 POKE USR "a"+2,120
40 POKE USR "a"+3,120
50 POKE USR "a"+4,60
60 POKE USR "a"+5,60
70 POKE USR "a"+6,24
80 POKE USR "a"+7,24
90 FOR r=15 TO 21
100 FOR c=0 TO 31
110 PRINT AT r,c;"■";
120 NEXT c: NEXT r
130 LET c=10
140 FOR r=0 TO 14
150 PRINT AT r,c;"▼"
160 PRINT AT r-1,c;" "
170 PAUSE 5
180 NEXT r
190 PRINT AT r,c-2;"▗▄▄▄▄▖"

0 OK, 0:1
```





INTRODUCING COLOUR

The Spectrum's television display is not limited to black letters and symbols on a white background. It can actually PRINT in any of eight colours (including black and white) and you can insert these colours in your programs. Each colour is identified by a number.

If you look at the number keys, you will see that keys 1–7 and 0 have a colour printed above them. To produce a colour on the screen, you need to use one of these colour codes together with a colour command.

Using colour commands

The Spectrum has three ways in which you can control colour. INK (on the X key) controls the colour of text PRINTed on the screen. PAPER (on the C key) selects the colour of the background, and BORDER (on the B key) controls the colour around the edge of the screen. To select a colour, you just have to key in one of these commands followed by a colour number.

To see what colours INK produces, key in this colour chart program:

INK COLOUR CHART PROGRAM

```
10 BORDER 0: PAPER 0: CLS
20 FOR i=1 TO 7
30 INK i
40 PRINT "INK ";i
50 PRINT "■■■■■■■■■■"
60 PRINT
70 NEXT i

0 OK, 0:1
```

Lines 20 to 70 contain a FOR ... NEXT loop which PRINTs a row of graphics characters (on the number keys) in each of the INK colours from 1 to 7. The program PRINTs all the colours except black:

INK COLOUR CHART



Line 10 makes the PAPER and BORDER areas turn black. You will notice that the line contains the command CLS. This is to ensure that the PAPER area is black before any text is PRINTed. Normally, the PAPER command only produces a coloured background immediately behind each PRINTed character. To colour the whole PAPER area, you must follow the PAPER command with CLS.

Changing INK, PAPER and BORDER

The next step is to see what happens when you use different INKs together with changing PAPER and BORDER commands. The following program does just this; it contains three FOR ... NEXT loops, one to control each of the colour commands. If you RUN it, you will see all the possible combinations of INK, PAPER and BORDER (there are 512 altogether!):

BORDER/PAPER/INK DEMONSTRATION PROGRAM

```
10 FOR b=0 TO 7
20 BORDER b
30 FOR p=0 TO 7
40 PAPER p: CLS
50 FOR i=0 TO 7
60 INK i
70 PRINT AT 1,11;"■■■■■■■■■■"
80 PRINT AT 3,11;"BORDER ";b
90 PRINT AT 5,12;"PAPER ";p
100 PRINT AT 7,13;"INK ";i
110 PRINT AT 9,11;"■■■■■■■■■■"
120 PAUSE 25
130 NEXT i: NEXT p: NEXT b

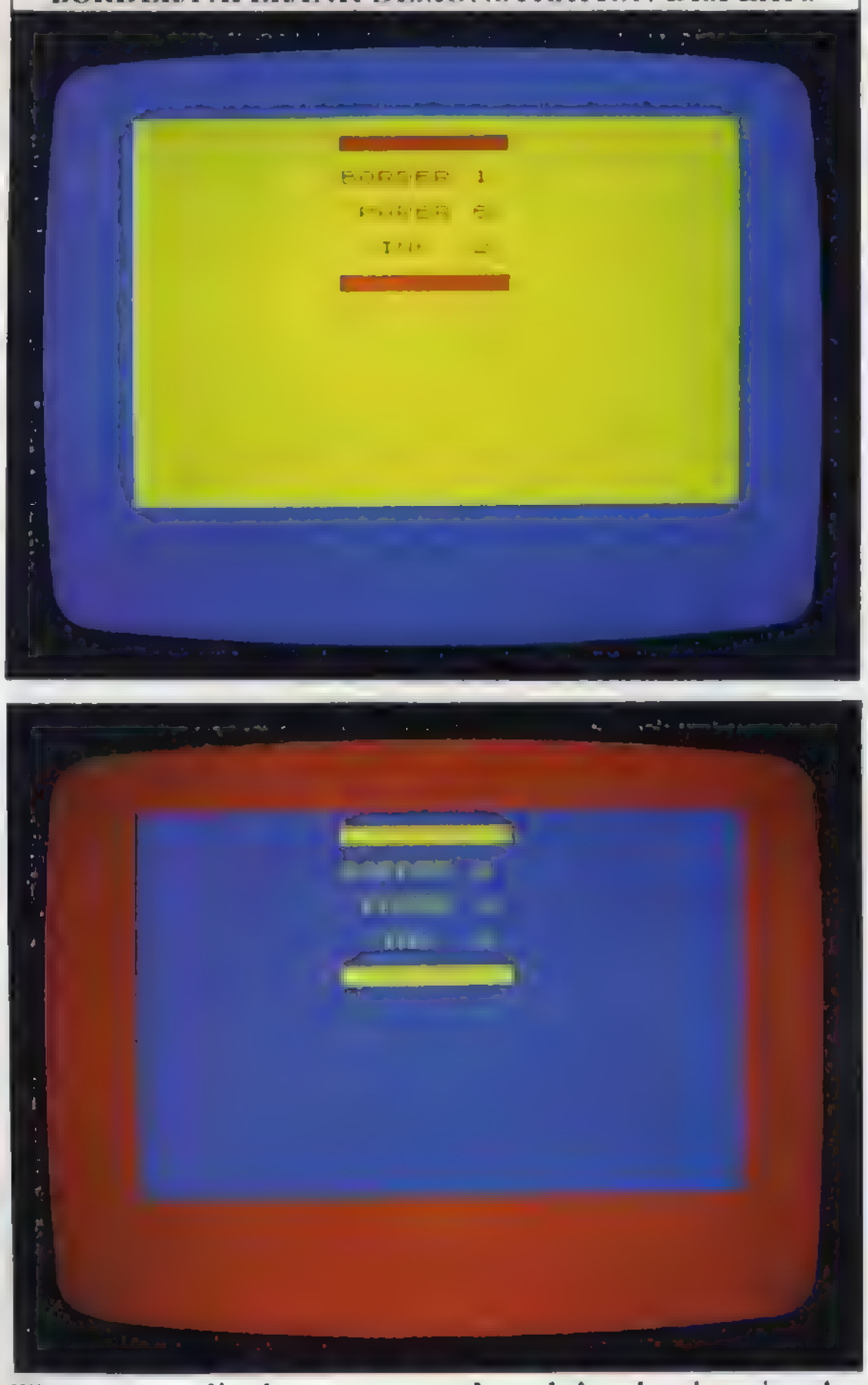
0 OK, 0:1
```

SPECTRUM COLOUR NUMBERS

Each colour command is followed by one colour number.

Number	Colour
0	Black
1	Blue
2	Red
3	Magenta
4	Green
5	Cyan
6	Yellow
7	White

BORDER/PAPER/INK DEMONSTRATION DISPLAYS



These two displays are produced by keying in the demonstration program. When you are writing your own program listings, you can work with any INK, PAPER and BORDER colours just by keying a line in (without a line number) before you start your listing. White on black, which produces an easy-to-read display, can be produced by keying in:

```
INK 7:PAPER 0:BORDER 0:CLS
```

Pressing NEW will reset the computer.

Improving the picture

If you are disappointed with the television picture your Spectrum is producing, you may be able to improve it by going through some simple checks. Make sure that the television is properly tuned into the computer's output signal. The tuning setting of both may drift from time to time, so it is a good idea to check your television tuning periodically.

If the picture is covered by a herringbone pattern, and you have checked that the television is properly tuned, make sure that there is nothing nearby that might be interfering with the computer's signal – a video recorder or another television set for example. Finally, the more colourful and brilliant a picture is, the more distorted it will seem to be. Try reducing colour, brightness or contrast to improve this.

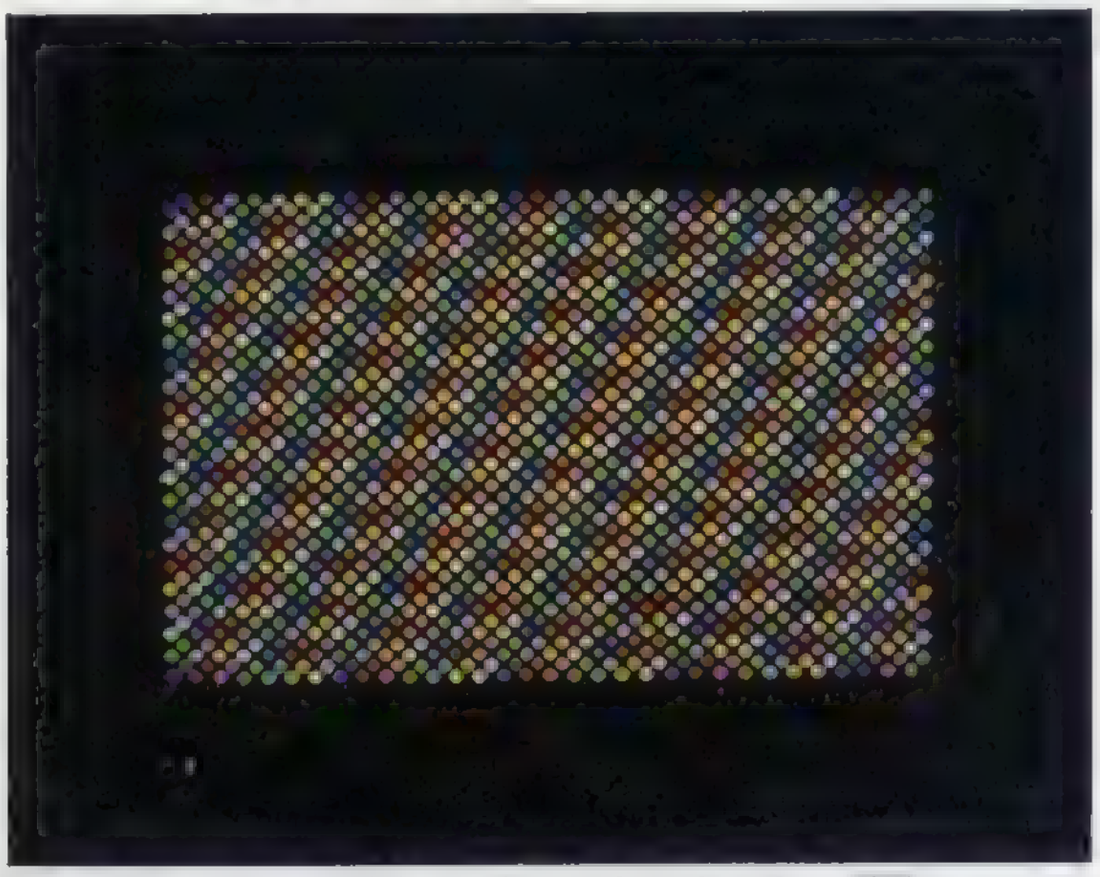
Colouring user-defined characters

If you define your own character with POKE USR (as on pages 30–31) you can PRINT this in any INK colour. Here is a program which produces a character – a pair of small stars – and which PRINTs it all over the screen in INK colours from 1 to 7:

COLOUR STARS PROGRAM

```
 10 BORDER 0: PAPER 0: CLS
 20 POKE USR "h",6
 30 POKE USR "h"+1,15
 40 POKE USR "h"+2,15
 50 POKE USR "h"+3,6
 60 POKE USR "h"+4,96
 70 POKE USR "h"+5,240
 80 POKE USR "h"+6,240
 90 POKE USR "h"+7,96
100 FOR n=1 TO 7
110 INK n
120 PRINT "■";
130 NEXT n
140 GO TO 100

0 OK, 0:1
```



Experimenting with colour is simple. When you become more adept at it you can begin to create new colours by optical effects. For instance make up user-defined characters from a grid of dots of which every other dot is a second foreground colour. Now, colour the background in a contrasting colour. Red dots on a blue background for example will mix to appear purple, while red and yellow will appear orange.

COLOUR GRAPHICS

Colour can be added to Spectrum graphics using the same commands and techniques as are used to colour text and user-defined characters. You don't need to learn special commands, nor to write a long program to produce simple colour graphics. Here, for example, is the arrowhead fall program that you used for animation on page 33. If you type in the program again, you can then add some colour to it:

ARROWHEAD FALL PROGRAM

```
 10 POKE USR "a",255
 20 POKE USR "a"+1,255
 30 POKE USR "a"+2,126
 40 POKE USR "a"+3,126
 50 POKE USR "a"+4,60
 60 POKE USR "a"+5,60
 70 POKE USR "a"+6,24
 80 POKE USR "a"+7,24
 90 FOR r=15 TO 21
100 FOR c=0 TO 31
110 PRINT AT r,c;"■";
120 NEXT c: NEXT r
130 LET c=10
140 FOR r=0 TO 14
150 PRINT AT r,c;"A"
160 PRINT AT r-1,c;" "
170 PAUSE 5
180 NEXT r
190 PRINT AT r,c-2;"▗▄▄▄▄▖"

0 OK, 0 1
```

Programming graphics colour

Now key in an extra line to the program:

85 BORDER 2:PAPER 1:INK 4:CLS

If you look at the colour codes on page 34, you should be able to tell what effect the new line will have. Now RUN the program. Here is the adapted listing containing the new line:

COLOUR ARROWHEAD FALL PROGRAM

```
 10 POKE USR "a",255
 20 POKE USR "a"+1,255
 30 POKE USR "a"+2,126
 40 POKE USR "a"+3,126
 50 POKE USR "a"+4,60
 60 POKE USR "a"+5,60
 70 POKE USR "a"+6,24
 80 POKE USR "a"+7,24
 85 BORDER 2: PAPER 1: INK 4: C
LS
 90 FOR r=15 TO 21
100 FOR c=0 TO 31
110 PRINT AT r,c;"■",
120 NEXT c: NEXT r
130 LET c=10
140 FOR r=0 TO 14
150 PRINT AT r,c,"A"
160 PRINT AT r-1,c;" "
170 PAUSE 5
180 NEXT r
190 PRINT AT r,c-2;"▗▄▄▄▄▖"

0 OK, 0 1
```

You should find that the adapted program now produces a red arrowhead, a blue sky, green ground and red BORDER. When the arrowhead reaches the ground, the graphics characters that show the impact are now PRINTed in green against blue – because line 190 is also affected by the INK and PAPER commands in line 85. Here is the coloured display that the adapted program produces after the arrowhead has completed its fall from the top of the PAPER area:

COLOUR ARROWHEAD FALL DISPLAY



Animated action in colour

You now have the basic expertise needed to write an animated colour graphics program. As an example of some simple colour graphics, here is a program which brings together everything that you have learned so far. It is built up from a number of separate blocks or "modules", most of which you should be able to follow:

LASER ATTACK PROGRAM

```
 10 DATA 2,244,46,31,31,46,244,
2
 20 DATA 8,16,58,254,254,58,16,
8
 30 DATA 146,84,84,254,124,254,
124,254
 40 FOR n=0 TO 7: READ x
 50 POKE USR "a"+n,x
 60 NEXT n
 70 FOR n=0 TO 7: READ x
 80 POKE USR "s"+n,x
 90 NEXT n
100 FOR n=0 TO 7: READ x
110 POKE USR "d"+n,x
120 NEXT n
130 BORDER 0: PAPER 1: INK 6: C
LS
140 FOR r=16 TO 21
150 FOR c=0 TO 31
160 PRINT INK 4;AT r,c,"■"
170 BEEP 0.01,14
180 NEXT c: NEXT r

scroll?
```

LASER ATTACK PROGRAM

```
 190 PRINT INK 2;AT 15,4;"■"
 200 LET r=8
 210 FOR c=0 TO 20
 220 PRINT AT r,c;"▶◀"
 230 BEEP 0.08,-2: BEEP 0.02,12
 240 PRINT AT r,c;"  "
 250 BEEP 0.02,8
 260 NEXT c
 270 PLOT 40,56: DRAW INK 7;128,
52
 280 BEEP 0.1,4
 290 DRAW INK 1;-128,-52
 300 FOR r=8 TO 15
 310 BEEP 0.02,320/r
 320 PRINT AT r,c;"▶◀"
 330 BEEP 0.08,420/r
 340 PRINT AT r,c;"  "
 350 BEEP 0.02,320/r
 360 LET c=c+1
 370 NEXT r
 380 PRINT INK 7;AT 15,27;"▗▄▖"
"

0 OK, 0:1
```

You will probably be puzzled by lines 10 to 30. These lines are actually part of a quick way of producing user-defined characters. If each of the a, s and d keys were reprogrammed by the method we've been using up to now, it would take 24 lines – 8 lines of POKE USR statements for each letter. Using this new method, each key can be reprogrammed in a maximum of four lines, with a list of grid totals stored after each DATA command. The program uses three characters; two to make the front and back of a rocket, and another one to make a laser base.

The READ...DATA routine, explained on pages 50–53, is a quick method of assembling all the data you need to make up user-defined graphics. It enables you to put all the data into one statement, instead of using the POKE USR routine line by line.

Lines 40 to 120 then take the information that is stored in lines 10 to 30, and turn it into the characters themselves. The green ground is produced by lines 140–180, which contain a double FOR ... NEXT loop that repeatedly PRINTs a green square across the bottom of the screen. Line 190 then PRINTs the laser base.

Lines 200 to 260 control the movement of the rocket, sending it across the screen at a fixed height. You can see it on the top screen of the three in the right-hand column. You probably will not need to be told that the BEEP command produces the sound of the rocket. (You will shortly be coming on to using this command both for sound effects and notes).

Lines 270 to 300 program the laser to fire by DRAWing a line that hits the rocket (see the centre screen on the right). Once this has happened, lines 300 to 370 send the rocket plunging to the ground, while the final line PRINTs its crumpled wreckage. (In the final screen on the right the rocket is shown without the deletions which actually occur).

When you type in this program, remember that all the graphics symbols are obtained by switching to the graphics cursor (CAPS SHIFT +9) and pressing the appropriate user defined key – a, s or d. Remember also to remove the graphics cursor again before continuing.

LASER ATTACK DISPLAYS





SPECIAL SCREEN TECHNIQUES 1

As you saw on pages 32–33, it is possible to animate characters simply by PRINTing, erasing and rePRINTing the characters in a new position. If you want to take animation displays a step further, you can make use of two new Spectrum keywords INVERSE and OVER. These allow you to produce animation when the ordinary PRINTing technique will not work.

INVERSE is a command which switches over the INK and PAPER dot pattern on the screen, to give an effect which looks like the negative of a character. OVER lets you PRINT one character over another, so that the normal erasing that happens when you overPRINT does not take place.

INVERSE and OVER can be used to PRINT, PLOT or DRAW something over a background that itself has something DRAWn or PRINTed on it. This program shows what happens if you try to animate over a background without using these commands. It DRAWs a laser beam which fires across a grid:

LASER AND GRID PROGRAM

```
 10 BORDER 1: PAPER 7: INK 0: C
LS
 20 FOR y=16 TO 160 STEP 24
 30 PLOT 56,y: DRAW 144,0
 40 PLOT y+40,16: DRAW 0,144
 50 NEXT y
 60 PRINT INK 2;AT 20,1;"■"
 70 PRINT INK 2;AT 21,1;"■■"
 80 PLOT 16,16
 90 LET x=INT (RND*140): LET y=
INT (RND*160)
 100 DRAW INK 0;x,y
 110 DRAW INK 7;-x,-y
 120 GO TO 80

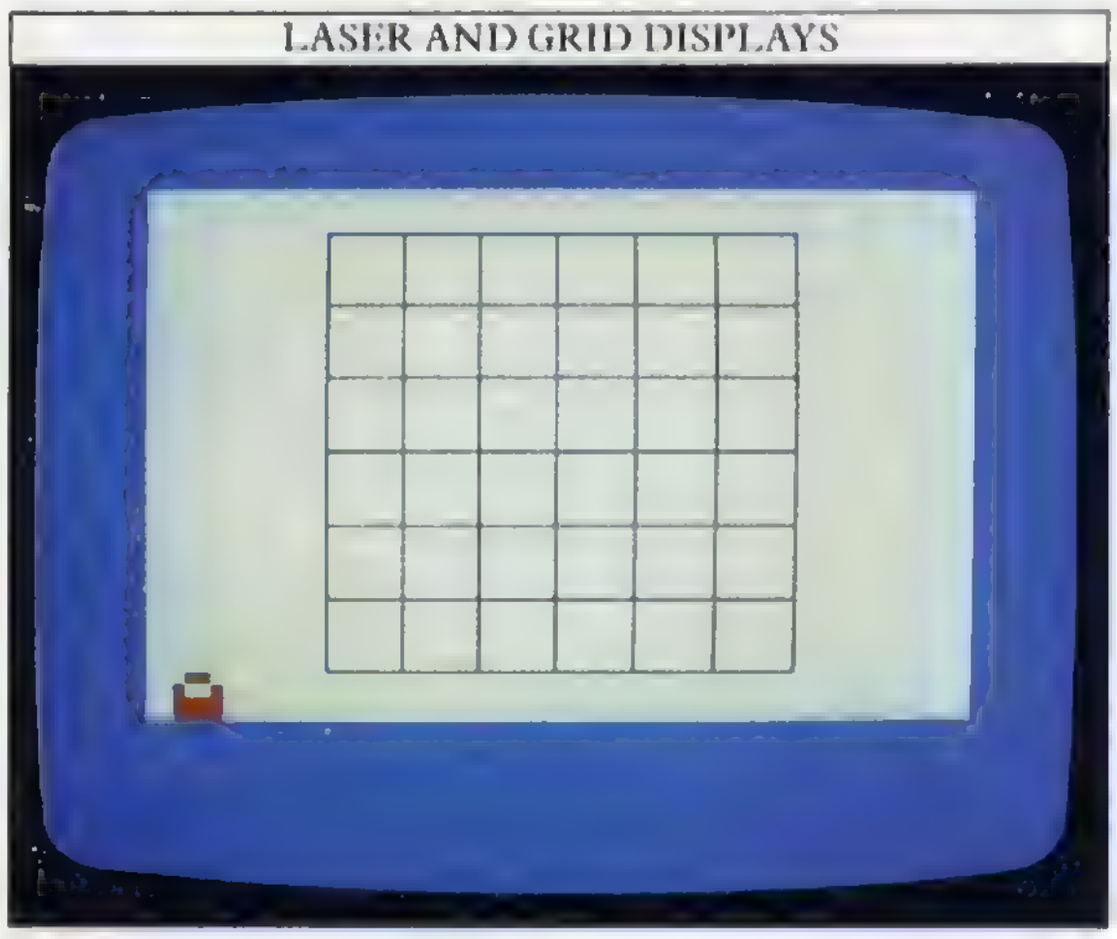

0 OK, 0:1
```

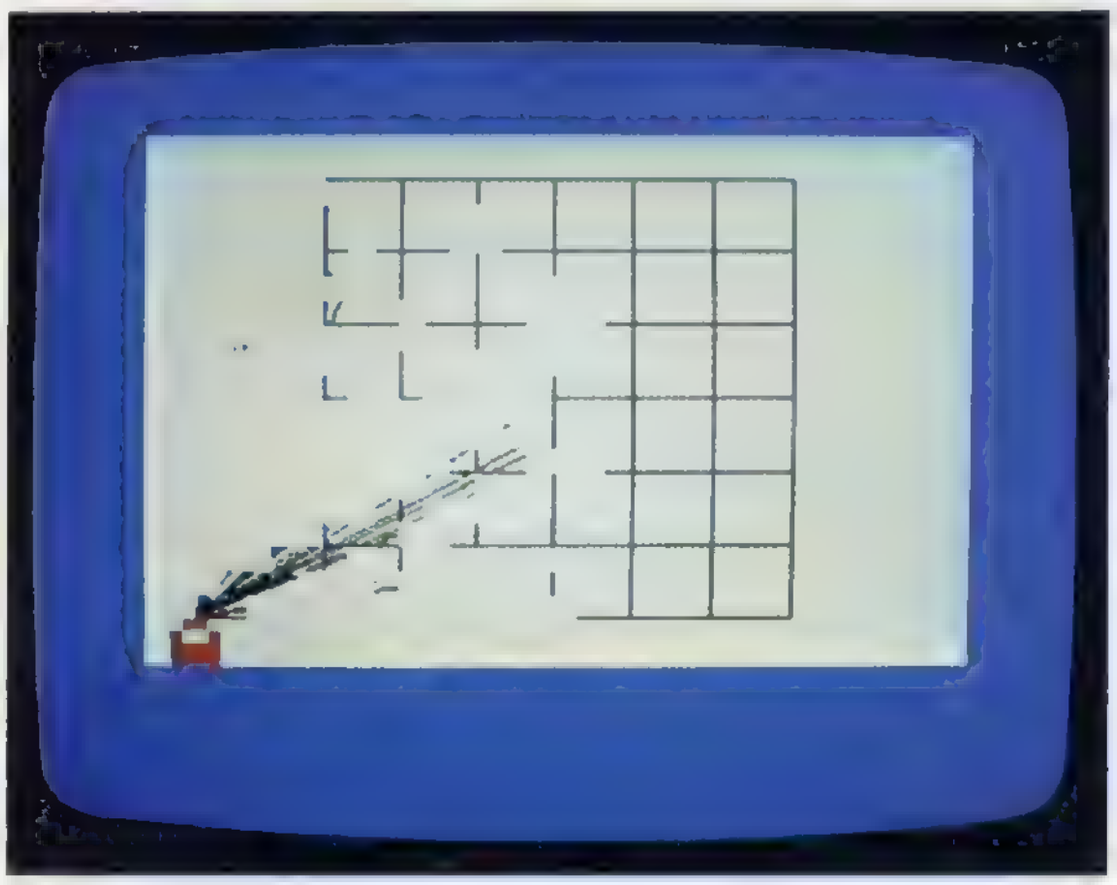
Lines 10 to 50 DRAW the grid – a black grid on a white background with a blue BORDER. Lines 60 and 70 PRINT a blue laser base in the bottom left corner of the screen. Line 80 PLOTs a point at the middle of the top of the laser base. This is simply a way of moving the graphics cursor to the top of the laser base, to the point where the laser beam will fire from. Line 90 produces values for x and y, which are the co-ordinates of the point to which the laser will fire – the end of the beam. The co-ordinates are set at random by the command RND (you will full details how to use this new on pages 48–49).

Line 100 DRAWs the black beam to x,y, while line 120 then "unDRAWs" the beam by DRAWing it in the background colour – white. Line 80 returns the program to the beginning of the firing routine again.

The next two screens show the display as it is when the program starts, and then as it is after the program has RUN for a while:

LASER AND GRID DISPLAYS





UnDRAWing and overPRINTing

You can see from the second display that the program doesn't do what is wanted. Firstly, when the beam is unDRAWn, the parts of the grid that lie along its path are also unDRAWn with it. Secondly, and perhaps more surprisingly, when each beam appears, so do the parts of all the previous beams that pass through character positions occupied by the current beam.

You could make the old beams invisible by adding INVERSE 1 to lines 100 and 110 like this:

```
100 DRAW INVERSE 1; INK 0;x,y
110 DRAW INVERSE 1; INK 0;-x,-y
```

The problem now is that, although the images of old

beams are eliminated, so is the current beam. Moreover, white squares still appear along the path of the beam, erasing parts of the black grid. You could try this instead:

```
100 DRAW INK 0;x,y
110 DRAW INVERSE 1;-x,-y
```

Now the beam will work properly, although its end point remains on the screen when the rest of the beam has been erased. The grid appears to be cut by white lines where the beam passes through it. The answer is to use OVER instead. Change lines 100 and 110 to:

```
100 DRAW OVER 1;INK 0;x,y
110 DRAW OVER 1;-x,-y
```

Now the beam works properly: it appears against the background of the grid and disappears again, without leaving any white cuts across the grid to mark its path. However, the end point of each beam remains on the screen. If this should fall on one of the grid lines, it leaves a white dot, breaking the line. This happens because even though the beam is DRAWn along one path and unDRAWn back down the same path back to its start point, the computer doesn't follow exactly the same path in both directions. So, to solve that, DRAW and unDRAW the beam along precisely the same path:

```
110 PLOT 16,16:DRAW OVER 1;x,y
```

Each beam is now DRAWn and unDRAWn from the same start point (16,16) to the same end point (x,y). The end points now no longer remain on the screen. The program works perfectly. The beams repeatedly flash across the grid and disappear again, leaving no evidence that they'd ever been there.

OverPRINTing with graphics

Now you can try replacing the laser beam with a larger graphics symbol, moving across a detailed background:

GRAPHICS WITH **OVER**

```
 10 BORDER 1: PAPER 1: CLS
 20 FOR r=15 TO 21
 30 FOR c=0 TO 31
 40 PRINT PAPER 4;AT r,c;" "
 50 NEXT c: NEXT r
 60 INK 0
 70 FOR x=0 TO 250 STEP 8
 80 PLOT x,55: DRAW 0,-55
 90 NEXT x
100 INK 6
110 FOR y=60 TO 170 STEP 10
120 PLOT 0,y: DRAW 255,0
130 NEXT y
140 LET r=13: LET c=3
150 PRINT OVER 1;AT r,c;"▄▄▄▄";
AT r+1,c+1;"▀ ▀"
160 PAUSE 5
170 PRINT OVER 1;AT r,c;"▄▄▄▄";
AT r+1,c+1;"▀ ▀"
180 LET r=r-1: LET c=c+1
190 IF r=0 THEN STOP
200 GO TO 150

0 OK, 0:1
```

Lines 20 to 50 PRINT a green ground on a blue background, which forms a blue sky. Lines 60 to 90 DRAW a series of black perspective lines on the green ground to help give the impression of depth. Lines 100 to 130 DRAW a series of yellow lines across the sky. Line 110 controls their spacing as they are DRAWn upwards on the screen. Lines 140 to the end of the program PRINT a small aircraft on the ground and then make it take off and fly up and out of the top of the screen. Using OVER, the background is left untouched. The moving symbol has no effect on the lines previously DRAWn:

OVER GRAPHICS DISPLAY



To get some experience of using INVERSE and OVER, try changing the colours in these programs, and substituting PAPER colours for INK colours and vice versa. Testing these commands in short programs will soon give you ideas for using them in graphics.

When you do use INVERSE or OVER, remember that they are commands which must be activated or de-activated by the number that follows them. On their own they will not work.

How to BRIGHTen up your displays

As well as producing displays in seven different "real" colours, the Spectrum can produce displays of different colour intensities. The command used to change brightness is simply BRIGHT. This is used in PRINT statements in the same way as INVERSE and OVER. To turn on extra brightness, you use BRIGHT 1, and to turn it off, you use BRIGHT 0. If you type in these lines:

```
10 BORDER 0:PAPER 0:CLS
20 PRINT "without BRIGHT"
30 PRINT BRIGHT 1;"with BRIGHT"
```

you will be able to see the difference that this command makes. You can also use BRIGHT 8. This enables you to PRINT BRIGHT text without altering the original PAPER colour in the position to be PRINTed on.

SPECIAL SCREEN TECHNIQUES 2

Many programs rely for part of their effect on characters that flash on and off. With the Spectrum, you could make characters appear to flash by rapidly changing the INK and PAPER colours. But don't rush to your keyboard to try this, because there is a much better and simpler way to achieve flashing. The Spectrum has a single command, FLASH, which produces the effect you want.

How to turn flashing on and off

FLASH can have one of two numerical values, 0 or 1. FLASH 1 makes a character flash, while FLASH 0 stops it again. The next program will show you what effect adding FLASH has. First, type in the program without FLASH:

PROGRAM WITHOUT **FLASH**



Line 10 sets up the screen with a blue BORDER and PAPER, and white INK. The range of x values in line 20 (1 TO 704) is chosen because there are 704 character positions on the television screen (22 lines of 32 characters). Now to make the display flash, add a new instruction to line 10 to make it read:

```
10 FLASH 1:BORDER 1:PAPER 1:INK 7:CLS
```

When you RUN the program this time, the INK-PAPER colours alternate, making the display flash. A screen full of flashing characters is rather difficult to look at, and, after a short time, you will probably want to stop it. If you type:

```
FLASH 0
```

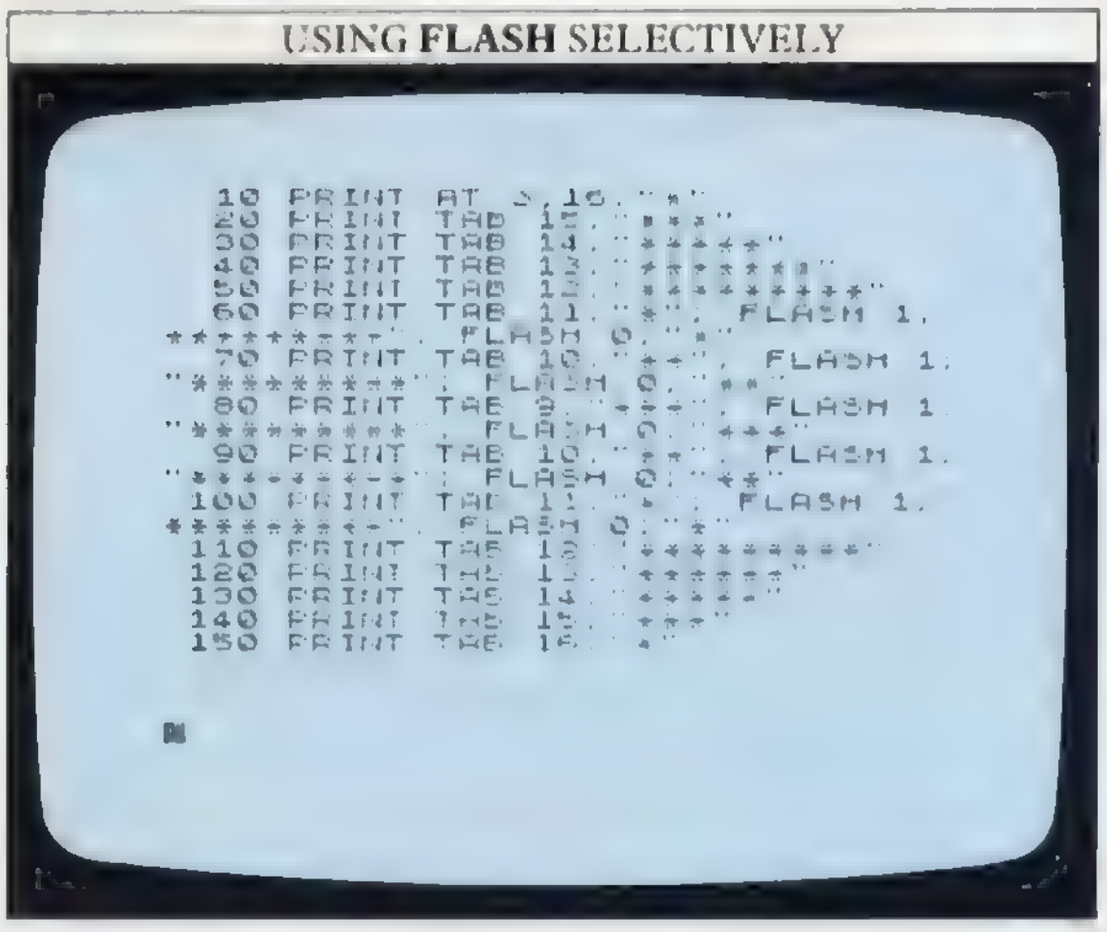
however, nothing happens. This is because the FLASH command only affects characters displayed after it. Instead, type CLS to remake the screen. The display should now stop flashing.

FLASH can be incorporated in PRINT statements in the same way as OVER and BRIGHT. All these commands can be used in one of two ways. If you now remove FLASH 1 from line 10 and key in:

```
30 PRINT FLASH 1;"#";
```

you will see the effect this has. Of course, you need not make the whole screen flash. In fact FLASH is a much more valuable command when used selectively. Try this display program:

USING **FLASH** SELECTIVELY



"Transparent" colour and contrast

Although the Spectrum has only eight screen colours, you may come across expressions using colour numbers 8 or 9. Colour number 8 produces a "transparent" colour. If you use PAPER 8 in a PRINT line, the PAPER colour will be left as it was before.

Colour number 9 does the exact opposite. If you've tried setting up your own colour screen, mixing the

available colours in different combinations, you will undoubtedly have discovered that some colours simply don't mix well. In some combinations, text disappears into the background, making it totally unreadable. INK 9 is very useful for achieving perfectly readable text first time. It PRINTs characters in either black or white, whichever contrasts more with the background PAPER colour:

INK 9 PROGRAM

```
 10 FOR p=0 TO 7
 20 INK 9: PAPER p: CLS
 30 PRINT AT 8,5;"INK 9 AUTOMAT
ICALLY"
 40 PRINT AT 10,5;"CONTRASTS IN
K COLOUR"
 50 PRINT AT 12,5;"WITH PAPER C
OLOUR"
 60 PAUSE 50
 70 NEXT p
```

Here a message is PRINTed against a background of each of the seven PAPER colours. The INK colour is chosen by the computer to contrast with the PAPER colour. White characters are PRINTed against black, blue, red and magenta backgrounds, but when the backgrounds change through green, cyan, yellow and white, the characters are PRINTed in black.

Overprinting without erasing

So far whenever you have PRINTed over text already on the screen, the original text has been blotted out by the new characters. However, this needn't always happen. The Spectrum keyword OVER which you met on the previous two pages works with text characters as well as graphics.

By using OVER, you can add characters together. For instance, lots of German words feature something called an *umlaut* – two dots above the letter a, o or u.

The Spectrum can generate accents, umlauts and double symbols with programs that overprint text like the first example in the three screens that follow. The same principle can be used to underline words on the screen, as you can see from the program in the centre screen in the next column. The underlining characters are PRINTed at exactly the same positions on the screen as one line of words. However, it doesn't, as you would normally expect, erase the words – it adds to them. You can see the result displayed on the bottom of the three screens. Try taking out line 20 and see the difference that it makes:

OVERPRINTING PROGRAMS

```
 10 PRINT AT 10,10;"Koln"
 20 PRINT OVER 1;AT 10,11;"¨¨¨¨"
 30 PRINT AT 12,10;"Societe"
 40 PRINT OVER 1;AT 12,14;"´´´"
 50 PRINT OVER 1;AT 12,16;"´´´"
 60 PRINT AT 14,10;"00000000"
 70 PRINT OVER 1;AT 14,10;"----
----"
```

```
0 OK, 0 1
```

```
 10 PAPER 0: BORDER 0: INK 7: C
LS
 20 OVER 1
 30 PRINT AT 8,6;"OVER LETS YOU
"
 40 PRINT AT 10,10;"UNDERLINE"
 50 PRINT AT 10,10;"_________"
 60 PRINT AT 12,6;"WORDS ON THE
 SCREEN"
```

```
OVER LETS YOU
UNDERLINE
WORDS ON THE SCREEN

0 OK, 50:1
```

By using OVER, you can convert letters into graphics characters, create foreign alphabets or, in games and graphics programs, you can make a character appear on top of a background line.

SOUND, NOTES AND MUSIC

The Spectrum can produce a wide range of sounds, all under the control of a single command, BEEP. This is accompanied by two variables – d (the duration or length of the sound) and p (pitch). For instance:

BEEP 1,0

produces a beep one second long at a pitch of middle C. The pitch of sounds is measured relative to middle C – above middle C, p is positive, below p it is negative. The Spectrum's pitch values range from −60 to +69. Increasing the value of p by 1 increases the pitch of the sound by one semitone. A semitone is the change in pitch between, for example, C and C#. Although d represents the length of the sound in seconds, it is not restricted to a whole number; fractions of seconds are quite permissible.

As you can see from the table of p values below, you can increase the pitch of a sound by an octave by adding 12 to p. You can play the full range of Spectrum notes by RUNning the following program. It takes about 15 seconds to complete the scale:

PITCH SCALE PROGRAM

```
  10 PRINT AT 10,5;"**************
********"
  20 PRINT AT 16,5;"**************
********"
  30 FOR p=-60 TO 69
  40 PRINT AT 12,9;"PITCH NUMBER
 ";
  50 PRINT AT 14,12;" ";p;" "
  60 BEEP 0.1,p
  70 PAUSE 10
  80 NEXT p


0 OK, 0:1
```

PITCH VALUES

Pitch values range from −60 to 69. The values for six octaves grouped around middle C (0) are shown here.

Note	Pitch Value					
G#, Ab	−16	−4	8	20	32	44
G	−17	−5	7	19	31	43
F#, Gb	−18	−6	6	18	30	42
F	−19	−7	5	17	29	41
E	−20	−8	4	16	28	40
D#, Eb	−21	−9	3	15	27	39
D	−22	−10	2	14	26	38
C#, Db	−23	−11	1	13	25	37
C	−24	−12	0	12	24	36
B	−25	−13	−1	11	23	35
A#, Bb	−26	−14	−2	10	22	34
A	−27	−15	−3	9	21	33

A FOR ... NEXT loop in lines 30 to 80 goes through all the possible p values from −60 to 69, sounding each note for a tenth of a second, and lines 10 and 20 PRINT a frame around the pitch number PRINTed by line 50.

Measuring the Spectrum's speed with sound

The next program uses BEEP as a signal. If you want to find out how quickly your Spectrum works, you can write a program to make it carry out a long series of calculations, and then time how long it takes to complete the series. For an approximate timing, your watch will be accurate enough, and to get around the problem of having to look at your watch and the screen at the same time, you can use BEEP to mark the beginning and end of the program. Here is one way by which you can do it:

SPEED TEST PROGRAM

```
  10 PRINT AT 6,4;"SPECTRUM SPEE
D TEST"
  20 PAUSE 100
  30 LET t=0: BEEP 0,1.25
  40 PRINT AT 9,4;"calculation s
ubtotal"
  50 FOR c=1 TO 1000
  60 LET t=t+c
  70 PRINT AT 12,11;t
  80 NEXT c
  90 BEEP 0,1,30
 100 PRINT AT 9,4;"     FINAL TOT
AL
     "



0 OK, 0 1
```

```
      SPECTRUM SPEED TEST

          FINAL TOTAL

              500500


0 OK, 100 1
```

After the program title has appeared, lingered a while (determined by PAUSE 100), the timing period begins.

A message on the screen and a BEEP lasting 1/10th of a second tell you when the calculation – adding together all the numbers from 1 to 1000 – has started. Some seconds later, there is a second BEEP, again lasting 1/10th of a second but at a higher pitch, marking the end of the timing period. In that time, your Spectrum has not only performed 1000 calculations, but it has also PRINTed out 1000 results! Dividing the total time by 1000 will tell you how long each calculation and PRINTing takes.

Programming simple tunes

With the BEEP command, it is quite easy to get the Spectrum to play a simple tune. Here to start with is a short sequence of notes. You just enter a BEEP command for each separate note:

SIMPLE NOTE SEQUENCE

```
10 BEEP 0.5 ,0
20 BEEP 0.5 ,4
30 BEEP 0.5 ,7
40 BEEP 0.5 ,12
50 BEEP 0.5 ,7
60 BEEP 0.5 ,4
70 BEEP 0.5 ,0

0 OK, 0:1
```

As all the BEEPs are 0.5 seconds long, you could save a little typing time by using a variable, t, and setting it to 0.5 at the beginning of the program. The BEEP statements would then look like this:

NOTE SEQUENCE – VARIABLE DURATION

```
 5 LET t=0.5
10 BEEP t,0
20 BEEP t,4
30 BEEP t,7
40 BEEP t,12
50 BEEP t,7
60 BEEP t,4
70 BEEP t,0

0 OK, 0:1
```

The advantage of this is that you can then change the value of t at the beginning of the program, and all the timings will then alter in step.

Music on the Spectrum

If you want to write real music on the Spectrum, you will have to tackle the problem of timing. Musical notes can be of various lengths. If you call the duration of a crotchet d, then the durations of the other notes are related to it like this:
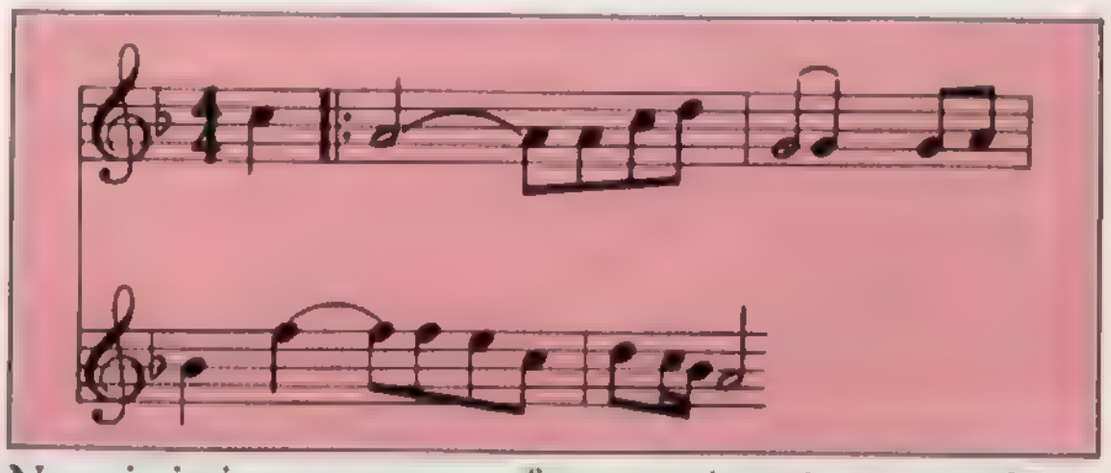
NOTE DURATIONS

If the duration of a crotchet is set at a variable d, the durations of other notes are as below.



Semi-quaver (sixteenth note)	d/4
Quaver (eighth note)	d/2
Crotchet (quarter note)	d
Minim (half note)	2*d
Semi-breve (whole note)	4*d

You can then set about converting a piece of sheet music into a computer program. Here are a few bars of a tune which you will probably recognize when you get your Spectrum to play them. The musical score for the first few bars looks like this:



Now it is just a matter of converting the notes. This listing sets d at 0.5; once you have entered the program you can vary the speed by altering this value. You will find that the pitch remains unaltered:

TUNE PROGRAM

```
  5 LET d=0.5
 10 BEEP d,12
 20 BEEP 2.5*d,9
 30 BEEP d/2,9
 40 BEEP d/2,12
 50 BEEP d/2,14
 60 BEEP 3*d,7
 70 BEEP d/2,7
 80 BEEP d/2,9
 90 BEEP d,10
100 BEEP 1.5*d,17
110 BEEP d/2,17
120 BEEP d/2,16
130 BEEP d/2,12
140 BEEP d/2,14
150 BEEP d/4,12
160 BEEP d/4,10
170 BEEP 2*d,9

0 OK, 0:1
```

SPECIAL EFFECTS WITH SOUND

So far you have just used the BEEP command to produce musical notes. However, when you come to writing your own programs, you will often want some quite unmusical sound effects to give the screen display added realism. Menacing sounds add a whole new dimension to many programs.

Writing sound loops

The simplest sound effect you can produce uses two BEEPs, at different pitches, and then cycles between the two by using GOTO to make a loop. Here is a program that produces a siren effect:

GOTO SOUND LOOP

```
10 BEEP 0.25,9
20 BEEP 0.25,7
30 GO TO 10

0 OK, 0:1
```

This simply plays a high note for a quarter of a second, followed by a lower note of the same length. You could write both BEEP statements on the same line so that the whole sound routine would take up only two lines:

```
10 BEEP 0.25,9:BEEP 0.25,7
20 GOTO 10
```

Because this program is written as an endless loop, you will need to press the CAPS SHIFT and BREAK keys to make it stop. However, by changing the loop to use FOR ... NEXT, you could incorporate it into a program as a sound effect that lasts for a set period of time, or that is repeated at intervals.

Altering the duration of a sound loop

The character of a sound can be changed dramatically if you shorten the playing time of the elements that make it up. By altering t in line 10 of either of the next two BEEP programs, you can hear the effect of shortening a sound sequence. These sorts of sound are often used in otherwise routine games to startle the player at a critical moment. The shorter the duration of the BEEPs, the more urgent the sound seems to become:

LOOP SOUND EFFECTS

```
 10 LET t=0.1
 20 FOR n=1 TO 20
 30 BEEP t,50: BEEP t,45: BEEP
t,40: BEEP t,45
 40 NEXT n
```

```
10 LET t=0.1
20 BEEP t,-25
30 BEEP t,-24
40 BEEP t,-23
50 BEEP t,-22
60 BEEP t,-21
70 GO TO 20

0 OK, 0:1
```

```
10 FOR n=1 TO 15
20 FOR p=60 TO 30 STEP -2
30 BEEP 0.005,p
40 NEXT p
50 NEXT n

0 OK, 0:1
```

The sound programmed in the third screen illustrates what happens when the BEEP gets really short. The program uses just one BEEP statement, but it features a technique that you haven't come across before – that of stepping backwards through a loop. The loop begins at p=60 and decreases p by 2 on each cycle until p=30. Each note sounds for only 0.005 (5 thousandths) of a second. If you make the note any shorter than this, it will sound like a click. You cannot produce a great variety of sounds with such simple sound commands. However, if you use sounds at the bottom end of the pitch scale, you can produce another type of effect. Try this program; it produces a chugging sound, rather like an engine:

LOW PITCH LOOP

```
10 FOR n=1 TO 100
20 FOR p=-30 TO -25
30 BEEP 0.01,p
40 NEXT p
50 NEXT n
```

```
0 OK, 0 1
```

Unpredictable sounds

So far, you have had a good idea of what sound a program would have produced before you ran it. But now try this program. (You will find the keyword RND above the T key):

RANDOM SOUND

```
10 LET p=INT (RND*60)
20 BEEP 0.01,p
30 GO TO 10
```

```
0 OK, 0 1
```

Instead of giving the pitch a fixed value or a predictable range of values, this program lets the computer choose pitches at random. (The use of RND is covered fully on pages 48–49).

Synchronized sound effects

It's relatively easy to generate sounds in isolation, as you have been doing so far, but incorporating sound effects in a program successfully is more difficult. The trick is to get the sound at the right duration at the right part of the program. To illustrate how this is done, the next program makes a simple graphics character move from one side of the screen to the other, while generating a sound effect in step with the movement:

SYNCHRONIZED SOUND AND ANIMATION

```
  10 BORDER 2: PAPER 6: INK 0: C
LS
  20 FOR c=0 TO 28
  30 LET r=11
  40 BEEP 0.02,12
  50 PRINT AT r,c+1;"▀"
  60 PRINT AT r+1,c+1;"▀"
  70 PRINT AT r+2,c;"▗ ▖"
  80 BEEP 0.2,0
  90 PRINT AT r,c+1;" "
 100 PRINT AT r+1,c+1;" "
 110 PRINT AT r+2,c;"  "
 120 BEEP 0.02,-5
 130 NEXT c
```

Instead of setting up user-defined graphics, the program uses the relatively coarse graphics available on the number keys. The character is composed of a total of eight separate symbols PRINTed over three lines. Normally, the character would be PRINTed and then, after a short time delay, erased before being PRINTed again one position further across the screen.

The delay makes sure that the character is on the screen longer than it is off, to minimize flickering. Here, though, the sound effect itself is used as a time delay. If one sound were made on each pass round the FOR ... NEXT loop (lines 20 to 130), it would be a rather disjointed, stuttering sound because of the relatively long silences. Splitting the sound effect into a number of different BEEP statements gets round that. It also makes more complicated sound effects possible than a single BEEP statement could produce. The lengths of the sounds are carefully chosen so that there is as little delay as possible between one character being erased and the next character being PRINTed. Since the BEEP command stops the program for as long as the sound is generated, it is necessary to display the character on the screen first and to generate the sound immediately afterwards.

DECISION-POINT PROGRAMMING

You have already looked at the concept of the loop in programs a number of times from page 26 onwards. If you want to carry out a calculation or put something on the screen 10 times, you could write:

```
FOR A=1 TO 10 ...
NEXT A
```

But there is another way of doing this, by using an IF ... THEN statement. To take an example, let's say you want to PRINT all the numbers from 1 to 10, together with their squares and cubes in a table. Here is how you would do it with FOR ... NEXT, and then with a different program which uses IF ... THEN:

FOR...NEXT LOOP

```
  10 BORDER 1: PAPER 6: INK 2: C
LS
  20 PRINT AT 4,6;"A";AT 4,14;"A
↑2";AT 4,24;"A↑3";AT 5,0;"_____
_____"
  30 FOR n=1 TO 10
  40 PRINT
  50 BEEP 0.1,40
  60 PRINT AT n+6,6;n;AT n+6,14;
n↑2;AT n+6,24;n↑3
  70 NEXT n

0 OK, 0:1
```



In the IF ... THEN program which follows, line 10 sets up the colour screen, and line 30 PRINTs the table's heading as before. Line 40 is the first line of the loop – it increases n by 1 on every pass round the loop. Line 70 is the same PRINT statement used in the FOR ... NEXT program. Line 80 is where the computer makes a decision as it examines n. The < symbol is mathematical shorthand for "less than". So, if n is less than 10, the computer is told to go around the program again from line 40. The computer continually PRINTs out A, A ↑ 2 and A ↑ 3 until n is not less than 10 when, in this case, it stops:

IF...THEN LOOP

```
  10 BORDER 1: PAPER 6: INK 2: C
LS
  20 LET n=0
  30 PRINT AT 4,6;"A",AT 4,14;"A
↑2";AT 4,24;"A↑3";AT 5,0;"_____
_____"
  40 LET n=n+1
  50 PRINT
  60 BEEP 0.1,40
  70 PRINT AT n+6,6;n;AT n+6,14;
n↑2;AT n+6,24;n↑3
  80 IF n<10 THEN GO TO 40

0 OK, 0:1
```

Why use the IF ... THEN loop?

You might wonder what the point of this is, as the IF ... THEN loop produces just the same results as the FOR ... NEXT loop. The value of IF ... THEN is that the computer can respond to any information that you INPUT during the program's operation by making a decision about it. Here is an example which shows this, by giving you a chance to test your skill at mental arithmetic (RND is explained on pages 48–49):

MATHS TEST PROGRAM

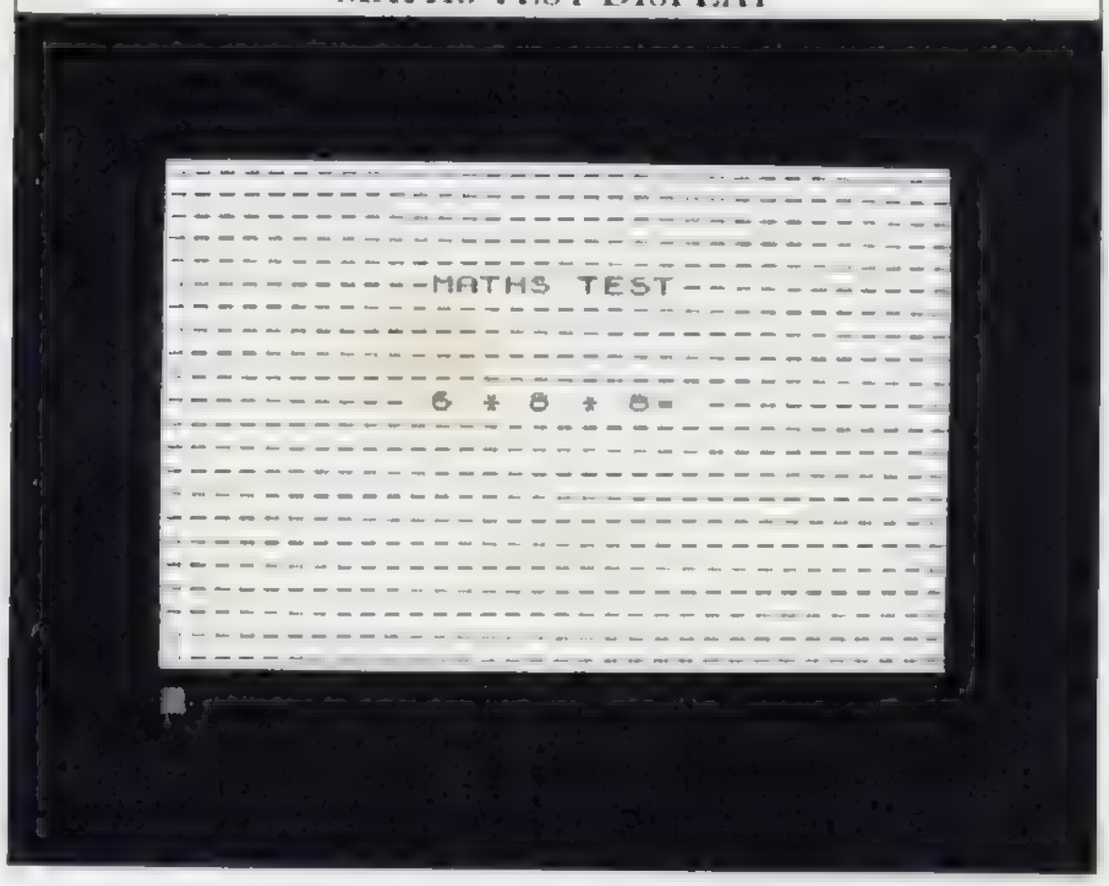
```
  10 BORDER 0: CLS
  20 FOR r=0 TO 21
  30 FOR c=0 TO 31
  40 PRINT AT r,c;"_"
  50 NEXT c: NEXT r
  60 PRINT AT 5,11;"MATHS TEST"
  70 LET x=INT (RND*10): LET y=I
NT (RND*10): LET z=INT (RND*10)
  80 PRINT AT 10,10;" ";x;" * ";
y;" * ";z;" = ": INPUT a
  90 IF a=x*y*z THEN GO TO 130
 100 BEEP 0.5,40: PRINT AT 10,10
;"---Urong------": PAUSE 50
 110 PRINT AT 10,10;"-Try again-
": PAUSE 50
 120 GO TO 80
 130 BEEP 0.5,50: PRINT AT 10,10
;"--Correct---": PAUSE 50
 140 GO TO 70

0 OK, 0:1
```

Each time the computer sets the problem and waits for your answer, it is faced with two possible courses of action. If you type in a correct answer, the IF ... THEN statement at line 90 directs the computer to go, not to line 100, but to line 130 next – PRINTing a "correct" message and then setting another problem. If the answer is wrong, then the computer "falls through" the IF ... THEN statement to line 100 and goes into the "wrong" routine.

It is important to remember that there must also be something in the program to stop the wrong answer routine carrying on into the correct answer routine. In this case, it is line 120, which makes the computer PRINT out the problem again:

MATHS TEST DISPLAY



Creating graphics with IF ... THEN

You can use IF ... THEN in combination with graphics commands to turn your Spectrum into an electronic drawing system. All you have to do is program the computer to respond to an INPUT by DRAWing:

IF ... THEN GRAPHICS PROGRAM

```
 10 BORDER 0: PAPER 0: INK 7: C
LS
 20 PLOT 25,25
 30 LET x=5
 40 INPUT k
 50 IF k=5 THEN DRAW -x,0
 60 IF k=6 THEN DRAW 0,-x
 70 IF k=7 THEN DRAW 0,x
 80 IF k=8 THEN DRAW x,0
 90 GO TO 40

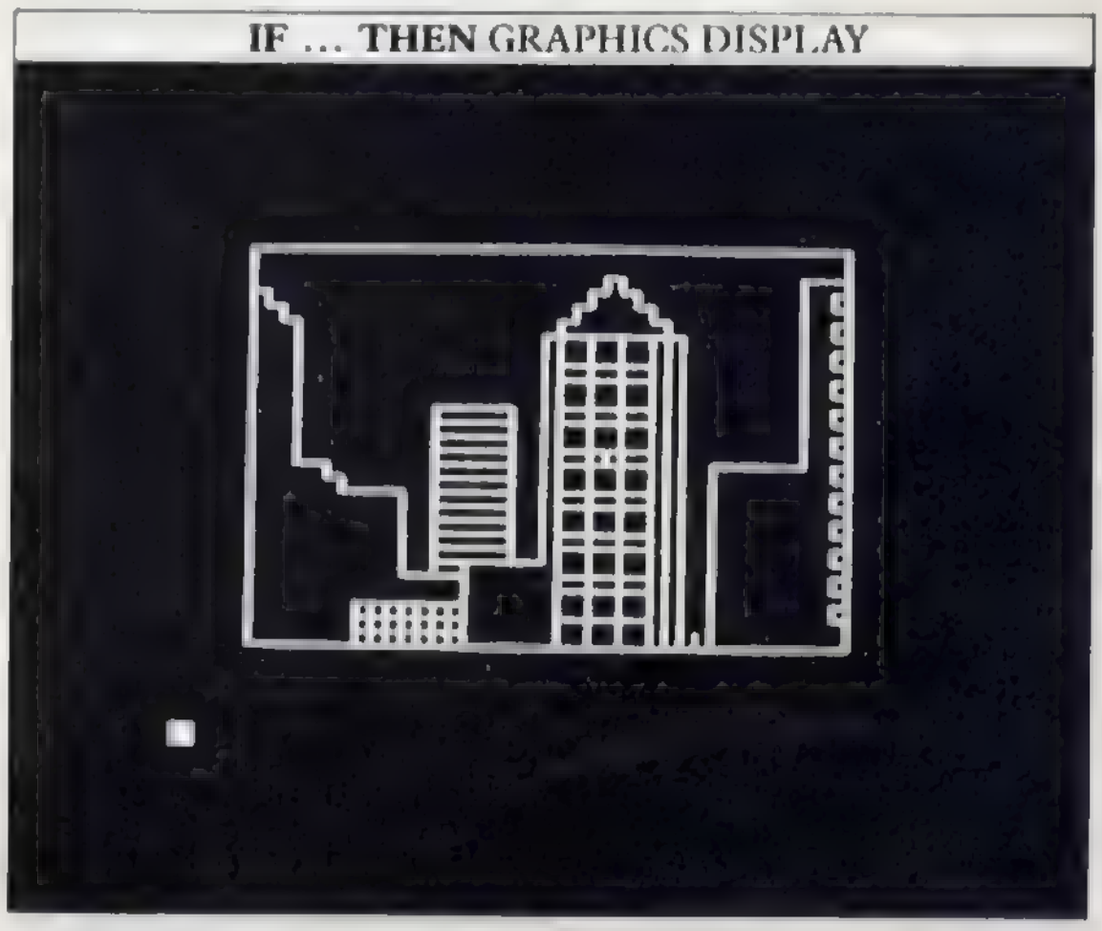

0 OK, 0:1
```

In this simple program the IF ... THEN statements allow the computer to decide what action to take. This program uses the cursor keys to DRAW lines either horizontally or vertically from a point near the bottom left of the screen.

The program simply PLOTs the point 25,25 and then DRAWs a small line every time you press one of the cursor keys. The four IF ... THEN lines make the computer examine your INPUT, and then decide in which direction the line should be DRAWn. Each time you use a key, remember to press ENTER so that the computer receives the instruction.

The statement in line 30 tells the computer how long to make each line. Making x equal to 5 gives quite good results, but you might like to try altering its value to change the resolution of your pictures. In addition to this, of course, you can change colour simply by altering the numbers in line 10. This screen will give you an idea of what you can DRAW:

IF ... THEN GRAPHICS DISPLAY



Selecting the right condition

Although you can extend this sort of program to use more keys, the Spectrum does have a better way of achieving the same effect, and having long rows of IF ... THEN statements is not really good programming. But when you use IF ... THEN, remember that there is a great variety of "conditions" which can follow the IF part of the statement. The programs on these pages have used either < or =, but this is only part of the complete range of symbols that the Spectrum uses as you can see from the following table. Choosing the right condition is not always easy, especially when you are dealing with moving characters.

IF ... THEN CONDITIONS

The symbols that follow the IF part of a line specify the kind of decision that the computer will make.

= is equal to	<> is not equal to
> is greater than	< is less than
>= is greater than or equal to	<= is less than or equal to

UNPREDICTABLE PROGRAMS

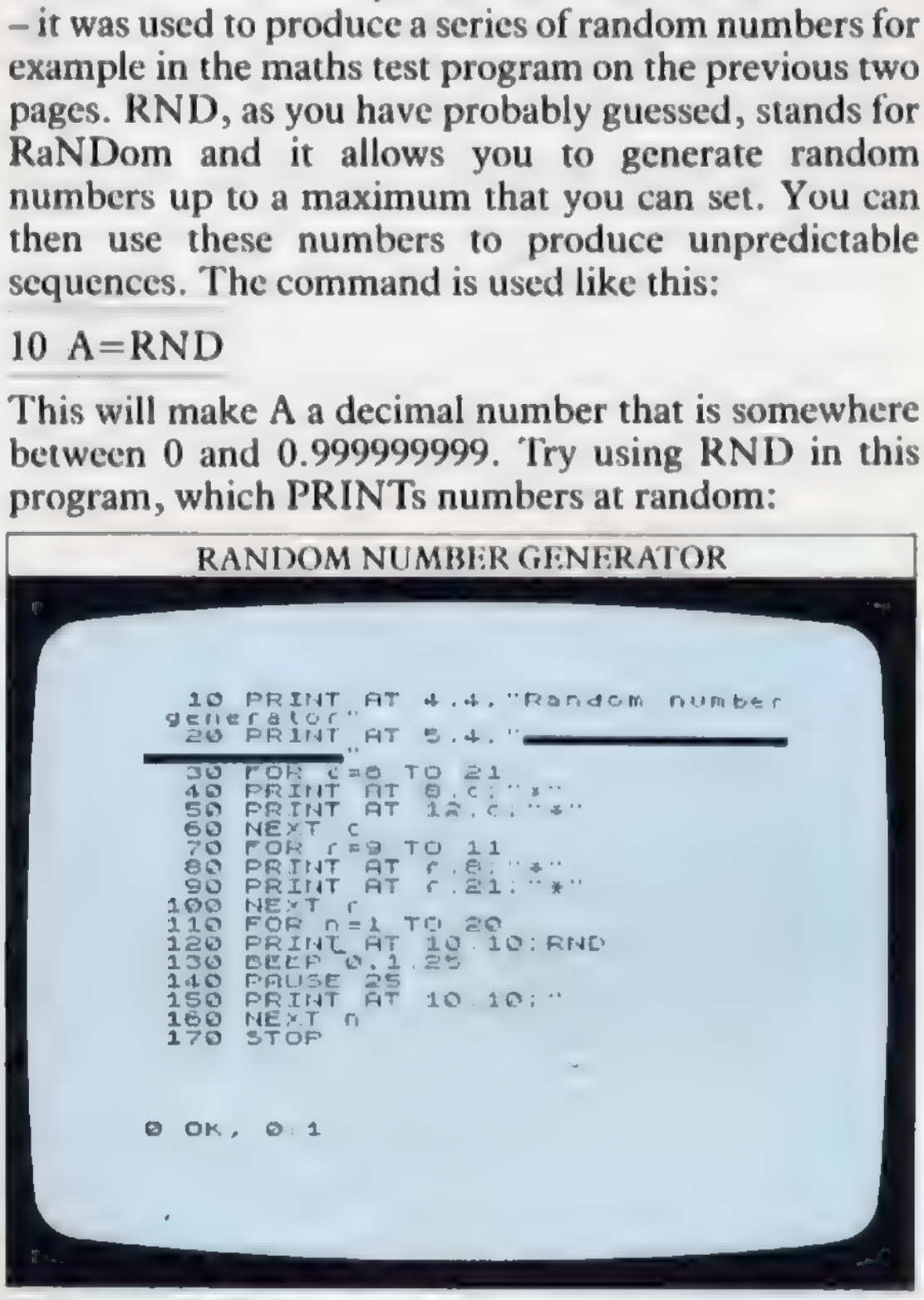
Although computers generally work with precise information, doing exactly what you tell them to do, an element of chance is necessary in certain applications. For instance, most computer games are based to some extent on luck. If you want to make something happen at an unpredictable time, or if dice are to be thrown or coins tossed, you can't tell the computer what result to produce every time or the element of chance would disappear.

The way to build chance into a program is to use RND. You will already have come across this command – it was used to produce a series of random numbers for example in the maths test program on the previous two pages. RND, as you have probably guessed, stands for RaNDom and it allows you to generate random numbers up to a maximum that you can set. You can then use these numbers to produce unpredictable sequences. The command is used like this:

```
10 A=RND
```

This will make A a decimal number that is somewhere between 0 and 0.999999999. Try using RND in this program, which PRINTs numbers at random:

RANDOM NUMBER GENERATOR



This uses RND in line 120 to generate random numbers between 0 and 0.999999999, while lines 30 to 100 set up a border of asterisks to frame the numbers. Very small numbers include the E symbol that you came across on page 17. Normally, as each new number is PRINTed, it automatically erases the last number – simply by PRINTing on top of it. However, when something like E-4 appears, it is not automatically erased, so the nine blank spaces in line 150 take care of that.

The Spectrum can generate only the limited range of random numbers used in this program. To generate other random numbers, you have to manipulate RND.

RANDOM NUMBER DISPLAY



Producing random whole numbers

If you now replace line 120 with:

```
120 PRINT AT 10,14;INT (RND*10)
```

and RUN the program again, you will notice an immediate change in the display. The numbers are no longer decimal fractions, in fact there's no decimal point at all. Instead, the program is generating whole numbers between 0 and 9 inclusive – a much more useful result for programs. INT, which you came across on page 27, rounds the random number produced down to the nearest whole number, or integer.

This way of using RND is very useful for programming games with chance built in. It is quite easy, for example, to get the Spectrum to simulate throwing dice or tossing coins:

COIN TOSS PROGRAM



As a tossed coin can only have one of two values – heads or tails – line 20 produces a random number that is either 1 or 2. Tails are represented by 1 and heads by 2. Two IF ... THEN lines determine what is to be PRINTed, and then the program continues:

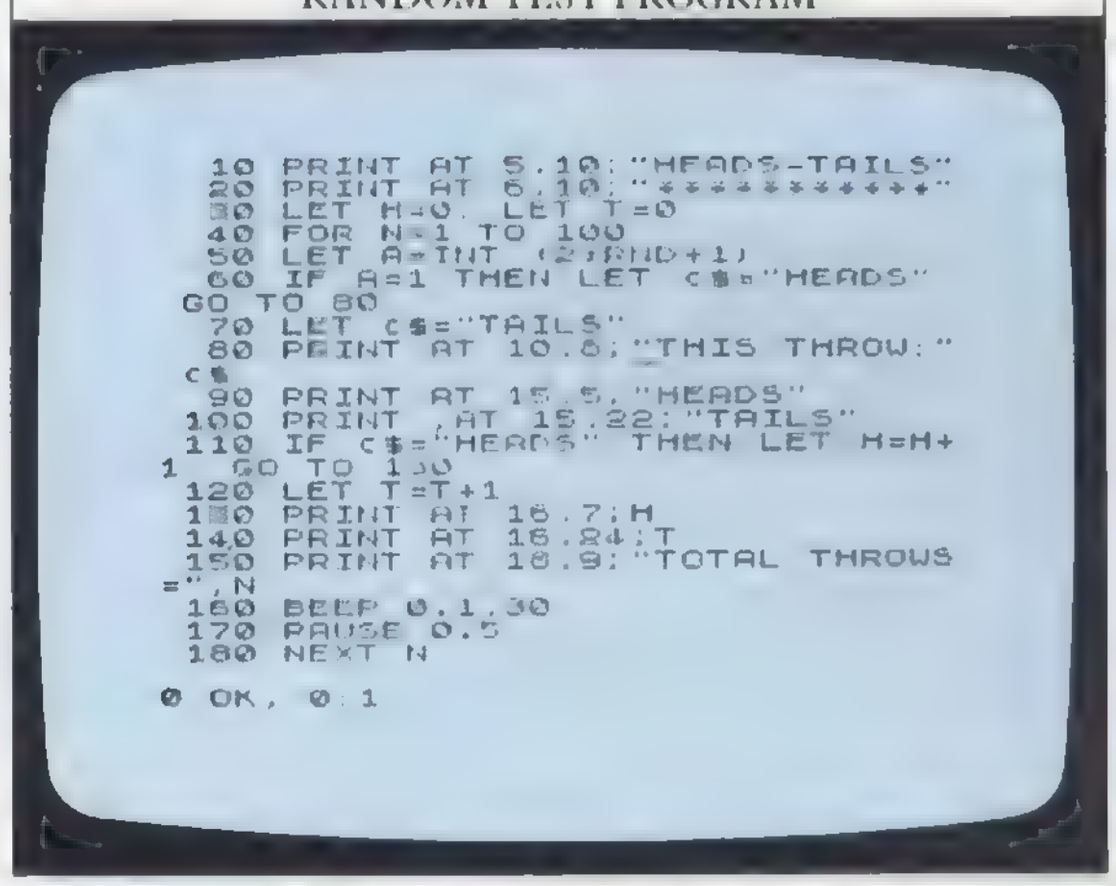
COIN TOSS DISPLAY



Checking a random sequence

It is possible to write a program that will show you just how random RND is. If you use RND to toss an electronic "coin" 100 times, you should get roughly 50 heads and 50 tails each RUN. You can actually test to see if this is true. Key in this program:

RANDOM TEST PROGRAM



When you RUN this, you will be able to see how close to 50:50 the heads and tails are each time the program is carried out.

Using RND in graphics programs

You can produce some interesting effects with RND by incorporating it in graphics programs, so that the computer is instructed to PRINT a character at a random position on the screen. If you then make the computer repeat this a number of times by using FOR...NEXT you can build up a display which will be different every time the program is RUN. Here is a program which uses RND in this way:

RANDOM GRAPHICS PROGRAM



Lines 10 to 80 define a character that is stored in the computer's memory and recalled by using the A key with the graphics cursor. Line 90 sets up a screen which is completely black – both BORDER and PAPER are set to 0. Lines 100 to 150 make up a FOR ... NEXT loop which selects a random position and a random INK colour, and which then PRINTs the character. PAUSE 0 (line 160) simply stops the program until any key is pressed, preventing the completion report from appearing:

RANDOM GRAPHICS DISPLAY



You can also use the RND command in programs to make keyboard characters, or even random points, appear in random colours. To do this, use the command INK c (where c is a random number between 1 and 7) in the same way as in line 140 of the program above.

COMPILING A DATA BANK

The data necessary for a program can be collected while it is RUNning by using INPUT, or alternatively can be written into the program itself. The commands used to store data are quite straightforward. Data is held in DATA statements and read by READ statements. Here is a program which will show you the technique at work. Type in:

CONSTELLATION PROGRAM

```
 10 BORDER 1  PAPER 1  INK 7: C
LS
 20 PRINT AT 0,11;"URSA MAJOR"
 30 PAUSE 50
 40 DATA 14,3,10,7,10,11,9,15,5
,24,10,25,13,20
 50 LET n=7
 60 FOR c=1 TO n
 70 READ x,y
 80 BEEP 0.05,25
 90 PRINT AT x,y;"*"
100 PAUSE 20
110 NEXT c


0 OK, 0:1
```

When you RUN this program you should see on your screen a computer-generated map of a group of stars, the constellation Ursa Major, also known as the Plough or Big Dipper:

CONSTELLATION DISPLAY



The information for the display is carried in line 40 in the form of 14 co-ordinates. Line 70 tells the computer to READ the DATA in line 40, and to understand the DATA as pairs of figures which the program will refer to as x,y. Line 50 tells the computer that there will be seven of these pairs altogether.

The computer first produces a short pulse of sound, and then PRINTs an asterisk AT each value of x,y, transforming the row of DATA into a map on the television screen.

With a program like this it is easy to enter new DATA to get the computer to PRINT a new map. Here is a set of line changes and the map it produces:

```
20 PRINT AT 0,11;"CASSIOPEIA"
40 DATA 8,3,12,8,9,14,14,18,8,24
50 LET n=5
```

CONSTELLATION DISPLAY



When you use DATA statements, it is important to tell the computer how much DATA there is to READ. Line 50 in the constellations program shows you how to do this. It sets the limit for the number of pairs of co-ordinates that are to be READ, so when the computer has PRINTed the final star, it stops. If there was no FOR ... NEXT loop in the second half of the program, the computer would run out of DATA. If this happened the program would end with an error report.

Storing numbers and strings together

Strings, too, can be stored and READ using DATA lines, and you can also store a mixture of both numbers and strings – the names of friends and their phone numbers or birthdays, for example. This does present a problem though, because two different types of READ statement are used to read numbers and strings – READ a and READ a$, for instance. But if you make all the DATA, including the numbers, appear as strings, you can overcome this difficulty.

The following program holds a personal telephone list. Names and telephone numbers are loaded by lines 10 to 50. Lines 60 to 80 display the program title and then offer a choice of functions:

TELEPHONE LIST PROGRAM

```
  10 DATA "C.Barnes","141","R.Bu
ckman","322","J.Hann","166"
  20 DATA "R.Harty","103","S.Ing
le","191","S.Jay","47"
  30 DATA "M.Kelly","33","M.Par
inson","86","M.Philbin","71"
  40 DATA "B.Redhead","122","M.R
odd","100","S.Scott","260"
  50 DATA "M.Stoppard","300","J.
Timpson","90"
  60 PRINT AT 5,4;"PERSONAL TELE
PHONE LIST"
  70 PAUSE 100
  80 PRINT AT 12,2;"COMPLETE LIS
TING....press 1";AT 15,2;"SELECT
IVE LISTING...press 2"  INPUT c
  90 CLS
 100 IF c=2 THEN GO TO 160
 110 PRINT  PRINT  PRINT
 120 FOR c=1 TO 14
 130 READ n$,m$
 140 PRINT TAB 6;n$;TAB 18;m$
scroll?
```

```
 150 NEXT c: PAUSE 0. CLS . GO T
O 60
 160 CLS . PRINT AT 9,5;"ENTER I
NITIAL AND NAME"
 170 INPUT e$
 180 LET r$="Name not found"
 190 RESTORE
 200 FOR c=1 TO 14
 210 READ n$,m$
 220 IF n$=e$ THEN LET r$=n$+"
     ..... "+m$
 230 NEXT c
 240 CLS : PRINT AT 10,5;r$
 250 PAUSE 200. CLS
 260 RESTORE . GO TO 60
0 OK, 0:1
```

To PRINT the whole telephone list, type:

1 then ENTER

COMPLETE TELEPHONE LIST

```
C.Barnes      141
R.Buckman     322
J.Hann        166
R.Harty       103
S.Ingle       191
S.Jay          47
M.Kelly        33
M.Parkinson    86
M.Philbin      71
B.Redhead     122
M.Rodd        100
S.Scott       260
M.Stoppard    300
J.Timpson      90
```

Alternatively, by typing in the following command:

2 then ENTER

you can find the number of just one name, after which the computer returns to the selection display:

TELEPHONE LIST SELECTION DISPLAY

```
PERSONAL TELEPHONE LIST

COMPLETE LISTING....Press 1

SELECTIVE LISTING...Press 2
```

If you type in 2 at line 80, the program follows lines 160 to 260. You are first asked to enter an initial and a name. Make sure that you PRINT in capitals and lower case as in the DATA lines, without spaces, or the computer will not recognize your entry.

If the computer finds that the name (e$) that you typed in is the same as one of the names (n$) in the DATA statements, it will give a new string (r$) the value of n$ plus a line of dots and the telephone number. If it does not find e$, r$ is left unchanged at "Name not found" (set by line 180), and that is PRINTed out at the end of the program.

Because you want to add the name, a line of dots and the telephone number together in line 220, the telephone number has to be treated as a string variable m$, instead of a numeric variable, m. If you used m the program would not work because string and numeric variables cannot be added together.

Lines 190 and 260 use a new command, RESTORE. This tells the computer to go back to the beginning of the DATA statements when it carries out the next READ command. Without it, the program would only RUN correctly once. This would be because the computer would run out of DATA; it would try to continue searching the DATA after the final item, but the program tells it to READ all the available DATA on each RUN. RESTORE lets the computer start searching from the beginning of the DATA each time as if it was the first RUN.

Once you know how to compile a DATA bank, you can use one to store your friends' birthdays, another to list bills and payments, or the details of your videotape or cassette library.

QUICK WAYS TO STORE CHARACTERS

On pages 30–31 you saw how to program and store your own graphics characters using the commands POKE and USR. As you will have noticed, getting the computer to produce even one user-defined character is quite a lengthy business. Programming a single graphics key takes eight program lines, so if you want to produce a symbol made up from four separate characters, you would need 24 program lines. However, the graphics programs on pages 36–37 used a short-cut that can make producing your own characters much faster.

Programming characters with READ ... DATA

As you saw on page 30, to program a graphics character you need to enter the numeric value of each line in the character grid. Here are two programs which produce the same character. The first uses a separate program line to store each separate total from the character grid:

SAVING LINES WITH **FOR...NEXT**

```
10 POKE USR "a",60
20 POKE USR "a"+1,126
30 POKE USR "a"+2,251
40 POKE USR "a"+3,254
50 POKE USR "a"+4,248
60 POKE USR "a"+5,255
70 POKE USR "a"+6,126
80 POKE USR "a"+7,60

0 OK, 0.1
```

```
 10 DATA 60,126,251,254,248,255
,126,60
 20 FOR f=0 TO 7
 30 READ x
 40 POKE USR "b"+f,x
 50 NEXT f

0 OK, 0.1
```

The second program instead stores the totals in a DATA line at the beginning of the program, and then converts them into the same character using a READ line.

You can see from this that READ ... DATA cuts the number of lines needed from eight to five. But although this is a useful saving, the great advantage of the READ ... DATA technique is that it actually saves you far more lines if you want to program a symbol that is made up from a number of characters, or if you want to use more than one symbol in a program. Then the savings become really worthwhile.

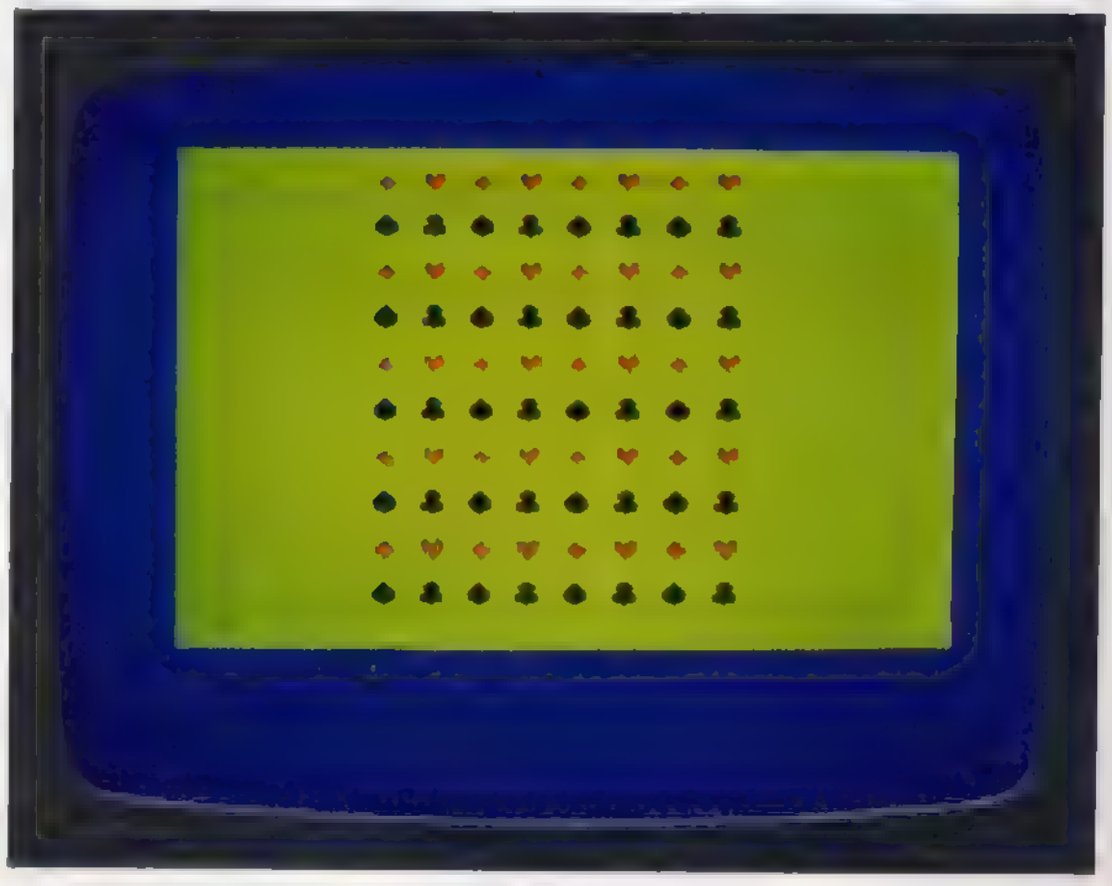
Storing groups of characters

Here is a program that reprograms four keys – a, b, c and d – to produce the symbols for four playing-card suits. Each of the symbols is produced by one loop. With a few extra lines you can create a display with the symbols:

READ...DATA WITH 4 LOOPS

```
 10 DATA 0,0,28,62,127,62,28,8
 20 DATA 34,119,127,127,127,62,
28,0
 30 DATA 8,28,62,127,127,127,62
,8
 40 DATA 28,62,62,28,127,127,12
7,0
 50 FOR f=0 TO 7
 60 READ x
 70 POKE USR "a"+f,x  NEXT f
 80 FOR f=0 TO 7
 90 READ x
100 POKE USR "b"+f,x  NEXT f
110 FOR f=0 TO 7
120 READ x
130 POKE USR "c"+f,x  NEXT f
140 FOR f=0 TO 7
150 READ x
160 POKE USR "d"+f,x  NEXT f

0 OK, 0.1
```



This way of producing the four symbols uses a total of 16 lines. This is eight lines less than the simple but laborious method which uses eight POKE ... USR statements per character. However, you can save even more space by READing all the DATA in the program in one FOR ... NEXT loop. Here is what the program would look like:

READ...DATA WITH 1 LOOP

```
  10 DATA 0,34,8,20,8,119,28,62
  20 DATA 28,127,62,62,62,127,12
7,28
  30 DATA 127,127,127,127,62,62,
127,127
  40 DATA 28,28,62,127,8,8,8,8
  50 FOR f=0 TO 7
  60 READ w,x,y,z
  70 POKE USR "a"+f,w
  80 POKE USR "b"+f,x
  90 POKE USR "c"+f,y
 100 POKE USR "d"+f,z
 110 NEXT f


0 OK, 0:1
```

At first sight, all the DATA seem to have changed. However, all that has really happened is that the DATA numbers have been keyed in a different order. The DATA is still held in the first four lines – 10 to 40. Line 60 tells the computer to READ this DATA in groups of four numbers, w,x,y,z, and each of these numbers reprograms a separate key. So the numbers that reprogram the A key are first, fifth, ninth, thirteenth, and so on. The key DATA are arranged like this:

DATA a,b,c,d,a,b,c,d,a,b,c,d ...

All you have to do to use this with your own symbols is to add up your grid totals, as in the diagram on page 30, and then make them into DATA lines by putting them together in the correct order. Now instead of saving eight lines of program, you will have saved thirteen.

How to use binary numbers with graphics

Unless you use a calculator, programming graphics characters can be quite a test of your ability at adding up. The code numbers for each column of squares in the character grid are not the most convenient numbers to deal with, and it is quite easy to make mistakes. But the Spectrum does let you key them in in another way, using the command BIN, which stands for BINary.

All computers use a binary system of electronic pulses to carry information. The word binary means that there are only two types of pulse – "on" or 1, and "off" or 0. All your programs are simply a stream of these electronic pulses. Every number that you type into the Spectrum is converted into a series of these "on" or "off" pulses, but because humans are not very familiar with the binary system, the computer is designed to accept "ordinary" numbers.

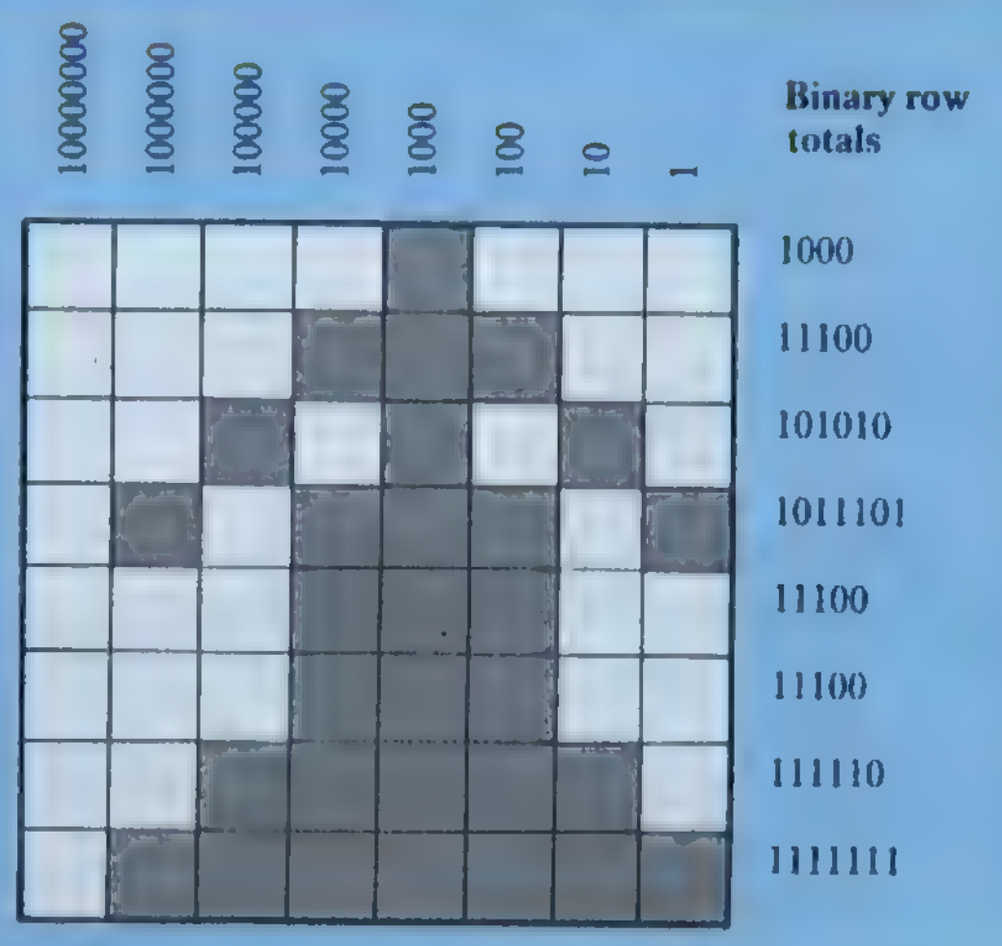
The command BIN by-passes this process and lets you enter binary numbers directly. BIN precedes a number that is written in multiples of two, instead of multiples of ten. Reset the computer and see what happens when you key in the following:

```
PRINT BIN 10
PRINT BIN 100
PRINT BIN 1011
```

Instead of seeing 10, 100 and 1011 on the screen, you should see 2, 4 and 11. The computer has converted the binary numbers back into "ordinary" numbers. The computer's ability to accept binary numbers is not of great value in simple programming. However, you can use the BIN command to save yourself having to do calculations when reprogramming keys.

You will remember that when you use a character grid, the numbers across the top of the grid go up in jumps. Each number is twice the one to its right. The computer actually gives each column a binary code, 0, 10, 100, 1000 and so on, so you can use binary numbers to enter the way a character is to be made up. Here is a grid numbered in this way:

USING A BINARY GRID



If you imagine that each filled-in square has a value of 1, and each empty square has a value of 0, you can enter the screen totals as sequences of ones and zeros using the BIN keyword or simply use BIN in a series of direct commands to PRINT each total in decimal. The great advantage of using BIN is that although the binary numbers look long, there is no adding up to be done. You simply key in what you see, counting black as 1, and white as 0.

ADVANCED COLOUR GRAPHICS

Having tried some simple colour graphics on pages 36–37, you can now move on to putting all the graphics, colour and animation commands together in one program to experiment with the ways they interact with one another. The program on these two pages produces a complex picture; if you work through this listing, you should be able to write a similar program yourself.

Creating a landscape

The first step in producing a display is to program it in black and white. Here is a listing that does that, using the "scrambled" DATA system that is dealt with on the previous two pages, and also using PLOT and DRAW to fill in two "pyramids":

PROGRAM BEFORE COLOURING

```
  10 DATA 144,9,127,254,73,146,6
3,252
  20 DATA 39,228,60,252,15,240,3
1,248
  30 DATA 31,248,15,240,63,252,3
9,228
  40 DATA 63,252,73,146,127,254,
144,9
  50 FOR n=0 TO 7
  60 READ p,q,r,s
  70 POKE USR "a"+n,p
  80 POKE USR "b"+n,q
  90 POKE USR "c"+n,r
 100 POKE USR "d"+n,s
 110 NEXT n
 120 FOR r=15 TO 21
 130 FOR c=0 TO 31
 140 PRINT AT r,c;"■"
 150 NEXT c: NEXT r
 160 FOR x=-24 TO 128
 170 PLOT 80,112
 180 DRAW x-80,-65
```

```
 190 NEXT x
 200 FOR x=72 TO 200
 210 PLOT 144,128
 220 DRAW x-144,-72
 230 NEXT x
 240 PRINT AT 1,27;"AB"
 250 PRINT AT 2,27;"CD"
```

```
0 OK, 0:1
```

The DATA lines (10–40) define four characters that are PRINTed together by lines 240 and 250 to produce the Sun. Lines 50 to 110 transform the information in the DATA lines into user-defined characters that are controlled by keys a, b, c and d. Lines 120 to 150 PRINT seven lines of the graphics character on the 8 key to form the foreground. Lines 160 to 230 then DRAW the two pyramids as simple triangles.

Once you have keyed in all the program, RUN it to make sure that you have typed it correctly. If the following display appears, you are ready to move on to the next stage:

DISPLAY BEFORE COLOURING



Programming colour and perspective

Now it is a simple matter to add the colour:

```
  5 BORDER 0:PAPER 1:CLS
140 PRINT INK 2;AT r,c,"■"
235 INK 6
```

When you RUN the program, the changes should result in the BORDER turning black and the PAPER blue, followed by a red foreground and yellow Sun.

You can now add the perspective lines, to give the display a feeling of "depth". The lines should produce the same effect as parallel paths disappearing into the distance. You need to DRAW lines between two horizontal rows of co-ordinates, with the top points being closer together than the bottom ones. Try keying in these lines:

```
260 LET w=8: INK 7
270 FOR x=72 TO 184 STEP 16
280 PLOT x,38
290 DRAW w-x,-38
300 LET w=w+34
310 NEXT x
```

The x co-ordinate of the top (most distant) end of each line is given by line 270. STEP 16 makes x increase by 16 each time instead of 1. The graphics cursor is then

moved to the top of the line by PLOT x,38. Each line is DRAWn from that point to the bottom of the screen by line 290. Then, to make the x co-ordinate of the bottom of the line (w–x) step across the screen in bigger steps than the top, line 300 increases w by 34. Once you have done this, RUN the program again:

"CHUNKY" GRAPHICS DISPLAY



As you will see, it produces odd results. The lines are DRAWn as large squares. What has gone wrong?

DRAWing lines on colour

The answer is that although the Spectrum's graphics use 256×176 pixels, INK colour resolution is only 32×22, the same as the text resolution. These large squares are known as "chunky graphics". As the lines are DRAWn, they automatically change the colour of every square they pass through. However, if you watch the top half of the screen as the picture forms, you will see that the lines used to DRAW the pyramids do not produce these chunky graphics. One further change is needed to create this complete non-chunky display:

CORRECTED DISPLAY



The problem is that lines can be DRAWn successfully on background PAPER colour but not on INK.

You must therefore rewrite the program so that the red ground is DRAWn as a background PAPER colour – not as INK. You can do this simply by changing line 140 to PRINT with red PAPER:

```
140  PRINT PAPER 2;AT r,c;"  "
```

Now the perspective lines will appear successfully.

Adding a moving character

For the final touch, you could add a bird flying across the screen. You can produce this by using two user-defined characters, and alternating them to simulate the bird's flapping wings:

DATA AND ANIMATION LINES

```
 320 DATA 0,16,0,16,0,40,2,58
 330 DATA 252,252,120,120,16,16,
0,0
 340 FOR n=0 TO 7
 350 READ t,v
 360 POKE USR "e"+n,t
 370 POKE USR "f"+n,v
 380 NEXT n
 390 FOR c=0 TO 31
 400 BEEP 0.02,40
 410 IF c/2=INT (c/2) THEN PRINT
 PAPER 1; INK 4;AT 4,c;"▼"
 420 IF c/2<>INT (c/2) THEN PRIN
T PAPER 1; INK 4;AT 4,c;"▼"
 430 BEEP 0.02,50: BEEP 0.02,45
 440 PRINT AT 4,c;" "
 450 NEXT c

0 OK, 0:1
```

Lines 410 and 420 use IF and THEN to make the computer check whether c, the bird's column position, is even or odd. If it is even, the bird's body alone is PRINTed, if it is odd, then a flapping wing is added. The display is now complete:

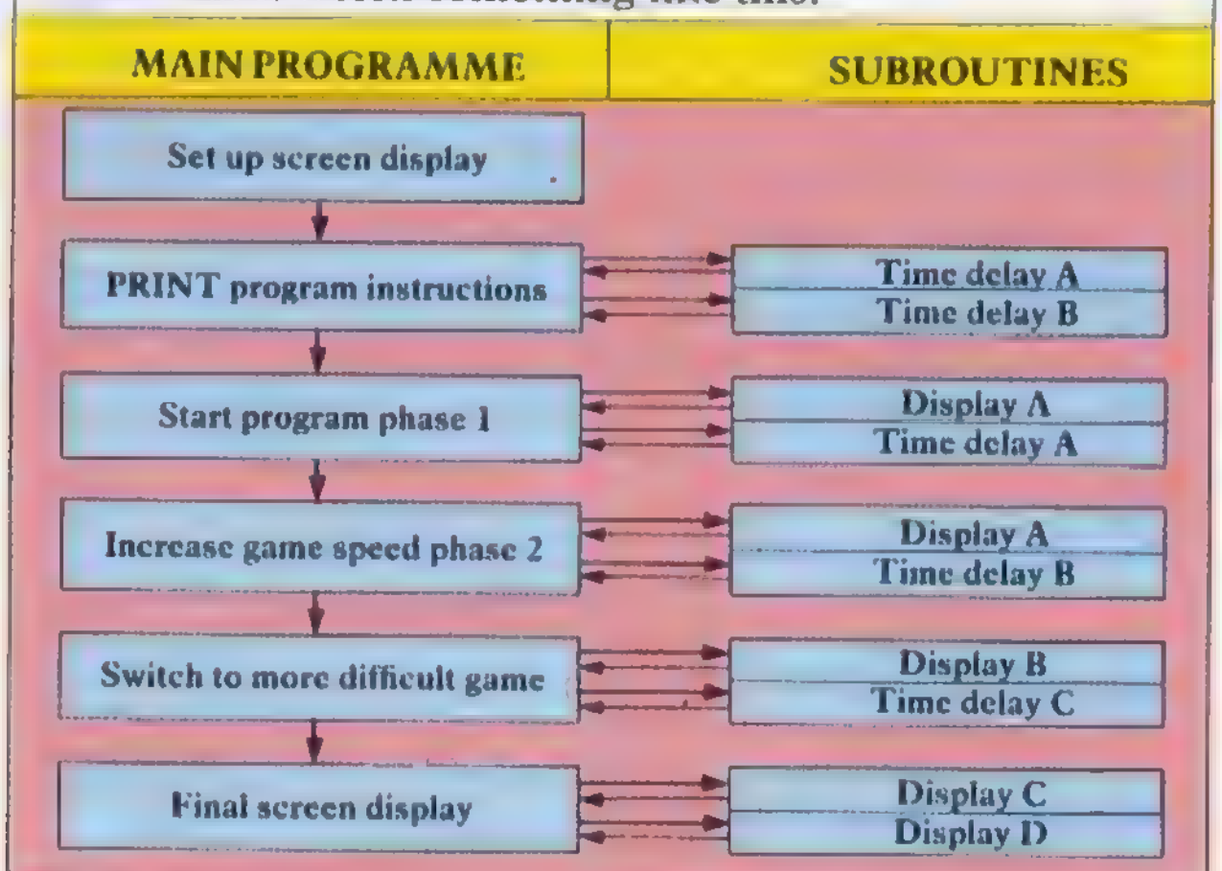
FINAL DISPLAY



WRITING SUBROUTINES

You will often want to use the same few lines of a program again and again to carry out the same calculation or to display the same group of characters on the screen. To avoid writing out the same lines time after time (and using up too much of the computer's memory) you could branch off to frequently-used sections of the program with GOTO. However, relying on GOTO is frowned on by many programmers. Using it carelessly can turn your programs into untidy mazes that are impossible to understand or debug.

The easiest program to analyze and debug is one that is written methodically in blocks or modules, each of which you can test independently of the others, if problems arise. If you look up the listing of a good games program in a magazine, for example, you will find that it works something like this:



How to use a subroutine

Frequently-used blocks of programs, or subroutines, are written using the command GOSUB. This allows you to branch off from the main program to the subroutine and then return to the main program again. The command looks like this:

```
50 GOSUB 500
```

Here the main program RUNs normally until it reaches line 50, which makes the computer jump to a subroutine at line 500. After it has been through the subroutine, it returns to the main program at line 60 – the one after the line where it left. The subroutine must be ended by the word RETURN. Without it, the computer will not go back to the correct point in the main program.

You can use GOSUB in almost any program where the computer has to repeat an operation. The next program produces a temperature conversion chart, using three types of measurement, Centigrade, Fahrenheit and Kelvin. The subroutine at line 80 makes the computer PRINT out a line on the table, produce a short BEEP, and then return to line 60. The command STOP at line 70 stops the program carrying on into the subroutine. If you miss out STOP, the computer will reach the RETURN command at line 110. It would then produce an error report because it had encountered a RETURN without its own GOSUB. You will notice that the subroutine in the listing below is inside a FOR ... NEXT loop, so it is "called" a number of times. The display produced when you RUN this program is shown on the bottom screen:

TEMPERATURE CONVERSION PROGRAM

```
  10 BORDER 1  PAPER 2  INK 7: C
LS
  20 PRINT AT 1,4;"C";AT 1,15;"F
";AT 1,26;"K"
  30 PRINT "-----------------------
----------"
  40 FOR c=-30 TO 140 STEP 10
  50 GO SUB 80
  60 NEXT c
  70 STOP
  80 PRINT TAB 3;c;TAB 14;(c*9/5
)+32;TAB 25;c+273
  90 BEEP 0.05,40
 100 PAUSE 25
 110 RETURN
```

```
0 OK, 0:1
```



In this program, the subroutine is not actually saving any space. However, if you extended the program to carry out other functions, the subroutine could be "called" again as often as you wanted – saving both space and memory.

Setting up displays with GOSUB

In many programs, you are initially given a "menu" or choice of options to select. This choice is often programmed by using GOSUB. When you enter your selection, the program goes to the appropriate subroutine and sets up the display you have picked.

Here is a simple listing that shows how you can do this. The program can set up either of two basic displays. One is illustrated on the bottom screen. The colours in each display are produced by a subroutine – which subroutine is used depends on your INPUT following line 20. If you were using this subroutine in a real games program, you could use these colour settings often by putting in a GOSUB:

MENU PROGRAM

```
 10 BORDER 1
 20 INPUT TAB 7;"DISPLAY 1 OR 2
?";a
 30 IF a=1 THEN GO SUB 300
 40 IF a=2 THEN GO SUB 400
 50 BORDER v: PAPER p: CLS
 60 PAPER q
 70 FOR r=15 TO 21
 80 FOR c=0 TO 31
 90 PRINT AT r,c;" "
100 NEXT c: NEXT r
110 INK 0
120 FOR y=0 TO 55 STEP 2.5
130 PLOT 0,y
140 DRAW 255,0
150 NEXT y
160 STOP
300 LET p=3: LET q=2: LET v=6
RETURN
400 LET p=1: LET q=4: LET v=2:
RETURN

0 OK, 0:1
```



Using GOSUB with animation

Here is a program which PRINTs a target – two graphics symbols on the ground – and which then PRINTs a succession of aliens falling from random points at the top of the screen. If one of the aliens then hits the target, line 240 directs the computer to go to the subroutine at line 270. The subroutine produces a sequence of changing BORDER colours while BEEPing, and then replaces the target, restarting the program. The bottom screen shows the display:

GOSUB ANIMATION PROGRAM

```
 10 DATA 0,0,64,32,16,11,13,31
 20 DATA 0,0,2,4,72,208,176,248
 30 DATA 59,25,15,63,79,95,129,
131
 40 DATA 220,152,240,252,242,25
0,129,193
 50 FOR n=0 TO 7: READ p
 60 POKE USR "a"+n,p
 70 NEXT n
 80 FOR n=0 TO 7: READ q
 90 POKE USR "b"+n,q
100 NEXT n
110 FOR n=0 TO 7: READ r
120 POKE USR "c"+n,r
130 NEXT n
140 FOR n=0 TO 7: READ s
150 POKE USR "d"+n,s
160 NEXT n
170 BORDER 1: PAPER 1: CLS
180 PRINT INK 2;AT 21,8;"cd"
190 LET x=2+INT (RND*15)
200 FOR f=0 TO 19

scroll?
```

```
210 PRINT AT f,x;"  "
220 PRINT INK 6;AT f+1,x;"AB"
230 PRINT INK 6;AT f+2,x;"CD"
240 IF f=18 AND x=8 THEN GO SUB
270
250 NEXT f
260 GO TO 180
270 FOR t=1 TO 30
280 BEEP .005,30+t
290 FOR g=0 TO 7
300 BORDER g: NEXT g
310 NEXT t
320 BORDER 1: LET f=19
330 CLS : RETURN



0 OK, 0:1
```



HINTS AND TIPS

When you are learning to program your Spectrum, you will have come across a number of ways of improving your technique by trial and error. However, there are some ways of saving time or sorting out problems which, although simple and effective, are not necessarily obvious. On these two pages you will find some "tricks" which will help you to produce programs that are well organized and bug-free.

Using REM as a marker or mask

Because the REM command makes the computer ignore anything that follows it in a line, it can be used in labelling and testing parts of a program. On page 18 you saw how REM can be used in the first line of a program to show you what a program does, and the longer a program is, the more useful this becomes.

However, when a program gets really long, it is sometimes difficult to pick out the REM lines among all the others. One way you can draw attention to them is by following REM with some symbols which clearly stand out from the rest of the program. Here is one way of doing this:

MARKER LINES WITH **REM**

```
  10 REM ************************
  20 REM COLOUR PYRAMID PROGRAM
  30 REM @ Ian Graham 1984
  40 REM ************************
  50 BORDER 0  PAPER 1: CLS
  60 DATA 144,9,127,254,73,146,6
3,252
  70 DATA 39,228,63,252,15,240,3
1,248
  80 DATA 31,248,15,240,63,252,3
9,228
  90 DATA 63,252,73,146,127,254,
144,9
 100 FOR n=0 TO 7
 110 READ p,q,r,s
 120 POKE USR "a"+n,p
 130 POKE USR "b"+n,q
 140 POKE USR "c"+n,r
 150 POKE USR "d"+n,s
 160 NEXT n
 170 FOR c=15 TO 21
 180 FOR c=0 TO 31

scroll?
```

When you read through this program, the REM lines are visible at a glance.

REM also has a use in program development. You will often want to test a program to see what happens if certain lines are left out. This may be because part of a program takes a long time to RUN, or produces a BEEP that you don't want to hear time and time again.

You can skip part of a program by using GOTO or RUN followed by a line number, but this won't help if you just want to miss out a few lines in the middle. The way to deal with this problem without deleting the lines is to insert a REM command at the beginning of each line you want to skip. This will mask or "disable" the lines, as the computer will ignore all the commands following each REM. Here is a program in which this has been done:

LINE DISABLED WITH **REM**

```
  10 BORDER 1: PAPER 2: INK 7: C
LS
  20 PRINT AT 1,4;"C";AT 1,15;"F
";AT 1,26;"K"
  30 PRINT "--------------------
----------"
  40 FOR c=-30 TO 140 STEP 10
  50 GO SUB 80
  60 NEXT c
  70 STOP
  80 PRINT TAB 3;c;TAB 14;(c*9/5
)+32;TAB 25;c+273
  90 REM BEEP 0.05,40
 100 PAUSE 25
 110 RETURN


0 OK, 0:1
```

How to check nested loops

When you use a number of loops in a program, it is easy to get the loops tangled so that the program does not produce the results you want. There is an easy way to check whether the loops are correctly "nested" – or that they fit inside each other without overlapping.

If you note the program down (or better still, if you can print it on a printer) you can connect the start of every loop up with its end:

LINKING LOOPS

```
  10 BORDER 0: PAPER 0: CLS
  20 FOR a=0 TO 20
  30 FOR c=0 TO 7
  40 PRINT INK c;"■";
  50 PAUSE 10
  60 NEXT c
  70 FOR c=0 TO 7
  80 PRINT INK c;"■";
  90 PAUSE 10
 100 NEXT c
 110 NEXT a
```

Alternatively you can jot down every FOR and NEXT in a program on a piece of paper in the order in which they appear, missing out the intervening lines. It's then an easy process to link the loops up.

If all the loops are correctly nested, none of these lines should overlap. If they do, you have wrongly nested loops, and the chances are that the program will not work correctly.

Improving keyboard feedback

Every time you press a key on the Spectrum, the computer makes a click. It's very useful. Without it, you have to keep looking at the screen to make sure that every key press has been registered by the computer. However, although it is loud enough for work in quiet surroundings, it can be drowned by noise from any other source, making programming more difficult. If the keyboard click isn't loud enough, you can increase it from a click to a more distinct BEEP by typing:

```
POKE 23609,n
```

where n is a whole number from 0 to 255. Zero produces a click, 255 a long, high-pitched note.

How to speed up editing

If you want to edit a line in the middle of a long program, you can simply type in a LIST instruction like this:

```
LIST 250
```

This will bring the line indicator down to the line to be edited. However, if the LISTing continues onto a further frame, the "scroll?" prompt appears at the bottom of the screen. If you press CAPS SHIFT and EDIT, the program will just scroll onwards to the next "page". You can press N to stop the listing scrolling and then carry on with editing, but there is an alternative that eliminates "scroll?" altogether.

If you want to edit line 250 in a long program, type:

```
249
```

followed by ENTER. Nothing much seems to happen. The line marker doesn't move. But more importantly, "scroll?" doesn't appear either. If you now press CAPS SHIFT and EDIT, line 250 then appears at the foot of the screen ready to be edited. The 249 – or whichever number you choose – should be a number which immediately precedes the number of the line that you want to edit. Make sure though that the number you use is not a line number already in the program. If it is, then typing the number followed by ENTER will erase the program line from the computer's memory.

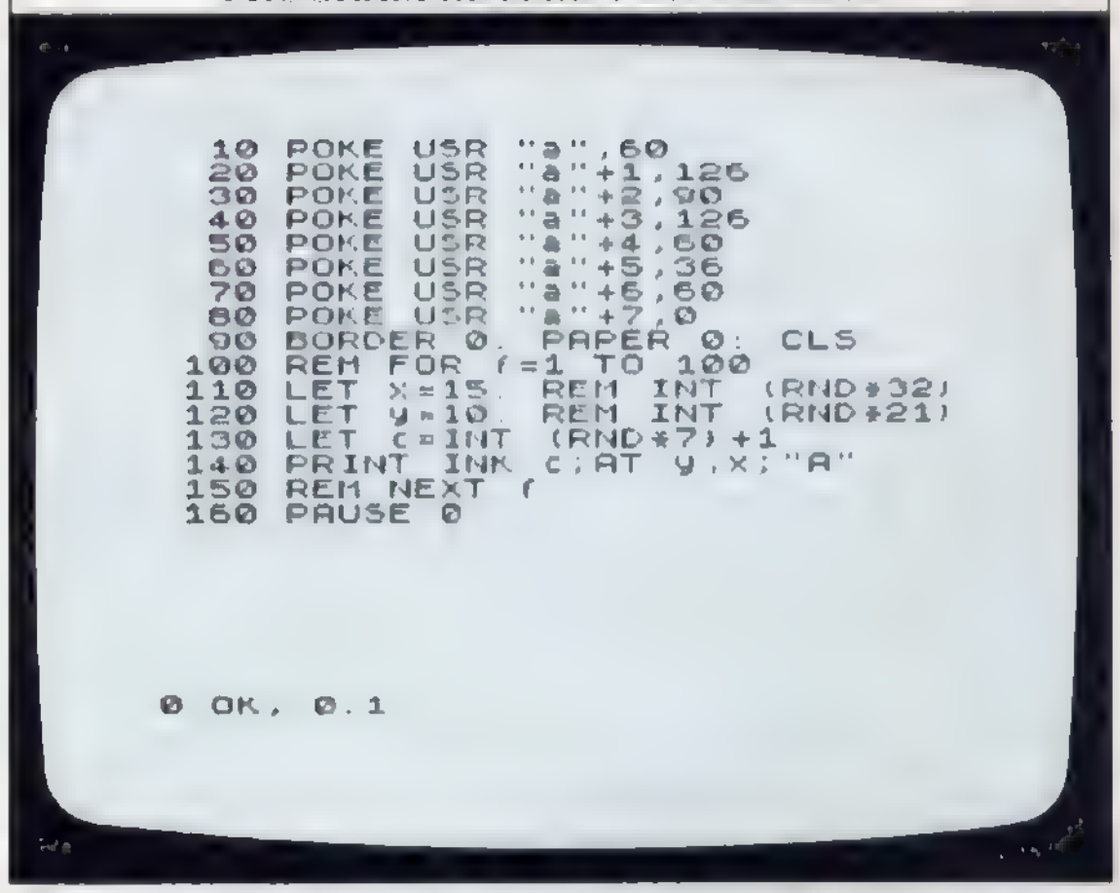
Useful debugging techniques

Although the Spectrum has a large repertoire of error reports which will alert you to any incorrect lines in a program, often a program will RUN without any hitches, only to produce a result entirely different to the one you had in mind. How then do you go about finding the source of the problem?

As you have just seen, you can use REMs to mask parts of a program, or you can link loops to check that they are nested properly. But if that doesn't help, you can often track down the problem by giving each variable in a program one set value, instead of allowing it to go through many.

Imagine that you have a graphics program which uses the command RND to produce a display which is built up by looping. If it does not work in the way you expect, you can take out the RND, and instead use a number. You can then work out what effect this number should have when the program is RUN. Now take out the lines that start and terminate the loop (you can use REM for this). If the result of a single RUN through is not what you predicted, the display should give you some idea of where your program is going "wrong":

PROGRAM EDITED FOR TESTING



Above is the random graphics program from page 49, edited so that the random variables in lines 110 and 120 are fixed. The original lines are still kept in, but are disabled by REMs. The loop between lines 100 and 150 is also disabled by a pair of REMs so the program only PRINTs once.

If the program is RUN, you can check whether or not the program has done what was expected, and if not, it is now much easier to work backwards to the source of the problem. You can use this technique in any program which uses variables. By substituting a single value for each variable, you can check your expected result with the result when the program is RUN.

Finally, don't forget that the BREAK key can be very helpful in telling you how far the computer has got through a program. If you RUN a program which either seems to do nothing, or gets stuck at a certain point, the BREAK key will tell you where the hold-up lies. If you then LIST the program, you will often be able to identify the problem with the line identified by BREAK and correct it.

SAVE
LOAD
VERIFY

HOW TO KEEP YOUR PROGRAMS

Whatever you type into your Spectrum is only stored in the computer's memory as long as it is supplied with power. When you switch the computer off, your program disappears. Obviously, you can't type in every program you want to use afresh each time you switch on the computer. Fortunately, you probably already have a means of storing programs cheaply and easily – an ordinary tape cassette recorder. The Spectrum has two sockets on its rear panel, labelled EAR and MIC, and these should be connected to the corresponding recorder sockets.

Setting the cassette controls

Having connected a recorder to the computer, the next job is to set the volume and tone controls properly. If there is a tone control, set it to maximum treble. The volume control may need a little more experimentation. Set the volume control midway between minimum and maximum.

Programs are recorded on tape and loaded back into the computer using two commands – SAVE and LOAD. You can test these commands by trying to SAVE any program from this book. Type the program into the computer again. RUN it to make sure that there are no typing errors and then unplug the EAR connection. Now give it any filename you like and ask the computer to SAVE it:

```
SAVE "FOR...NEXT"
```

The computer will reply with:

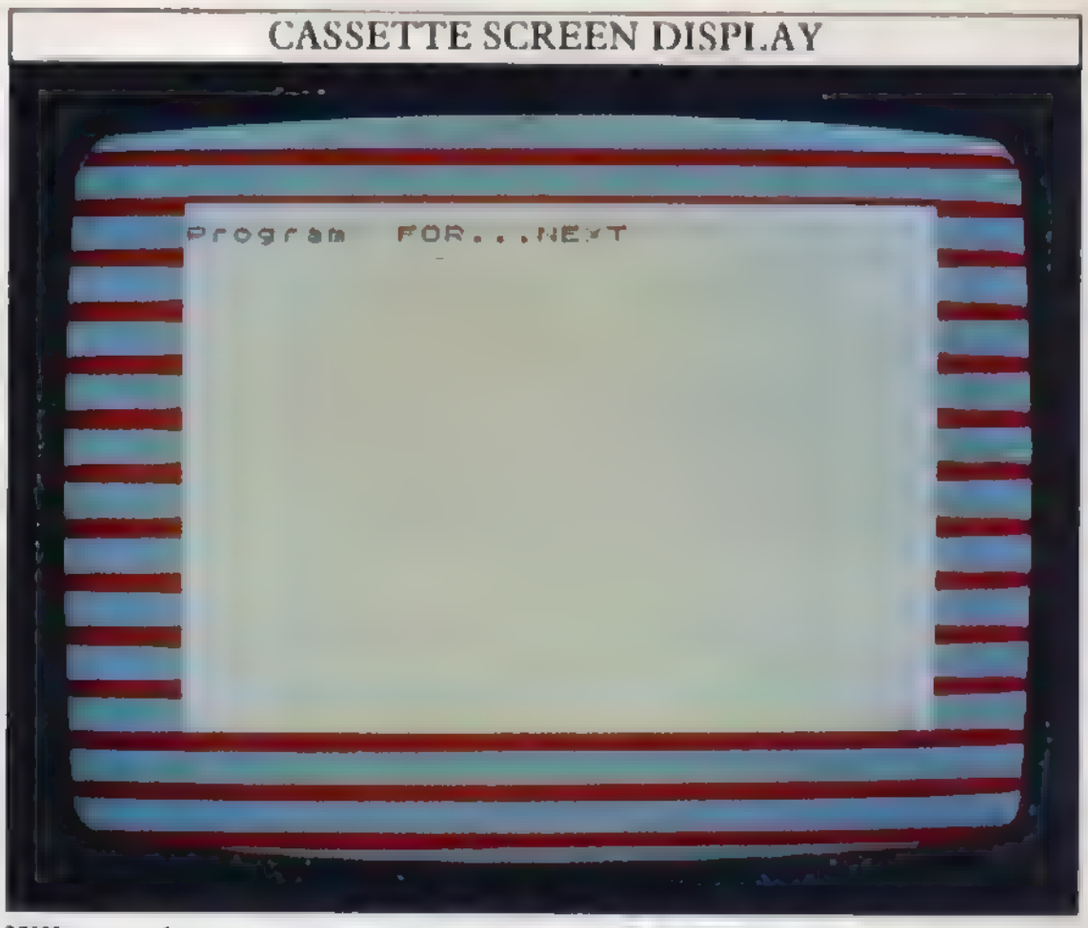
```
Start tape then press any key
```

Press RECORD and PLAY on the tape recorder and then press one of the computer keys. Two types of screen pattern will appear:

CASSETTE SCREEN DISPLAY



CASSETTE SCREEN DISPLAY



When the program has been SAVEd, the pattern of lines disappears and the OK prompt reappears at the bottom of the screen. Stop the tape.

Checking a recording

To check that the program has been recorded properly, reconnect the EAR lead and type:

```
VERIFY "FOR...NEXT"
```

Start the tape playing again. As each program recorded on the tape is played back into the computer, the pattern of red and blue stripes should reappear on the screen, followed by the program's name. If not, then the program has not been SAVEd properly. Rewind the tape, turn up the volume and press PLAY. If you still cannot VERIFY the program, interrupt VERIFY by pressing CAPS SHIFT plus BREAK. Try recording the program again at a higher volume setting (don't forget to remove the EAR plug).

You can use VERIFY to catalogue all the programs on a tape. Type VERIFY "cats" (or any other filename that you know you have not used). As each program on the tape is read by the computer, its filename will appear on the screen.

Playing back a program

Now try LOADing the program back into memory. Type in the filename like this:

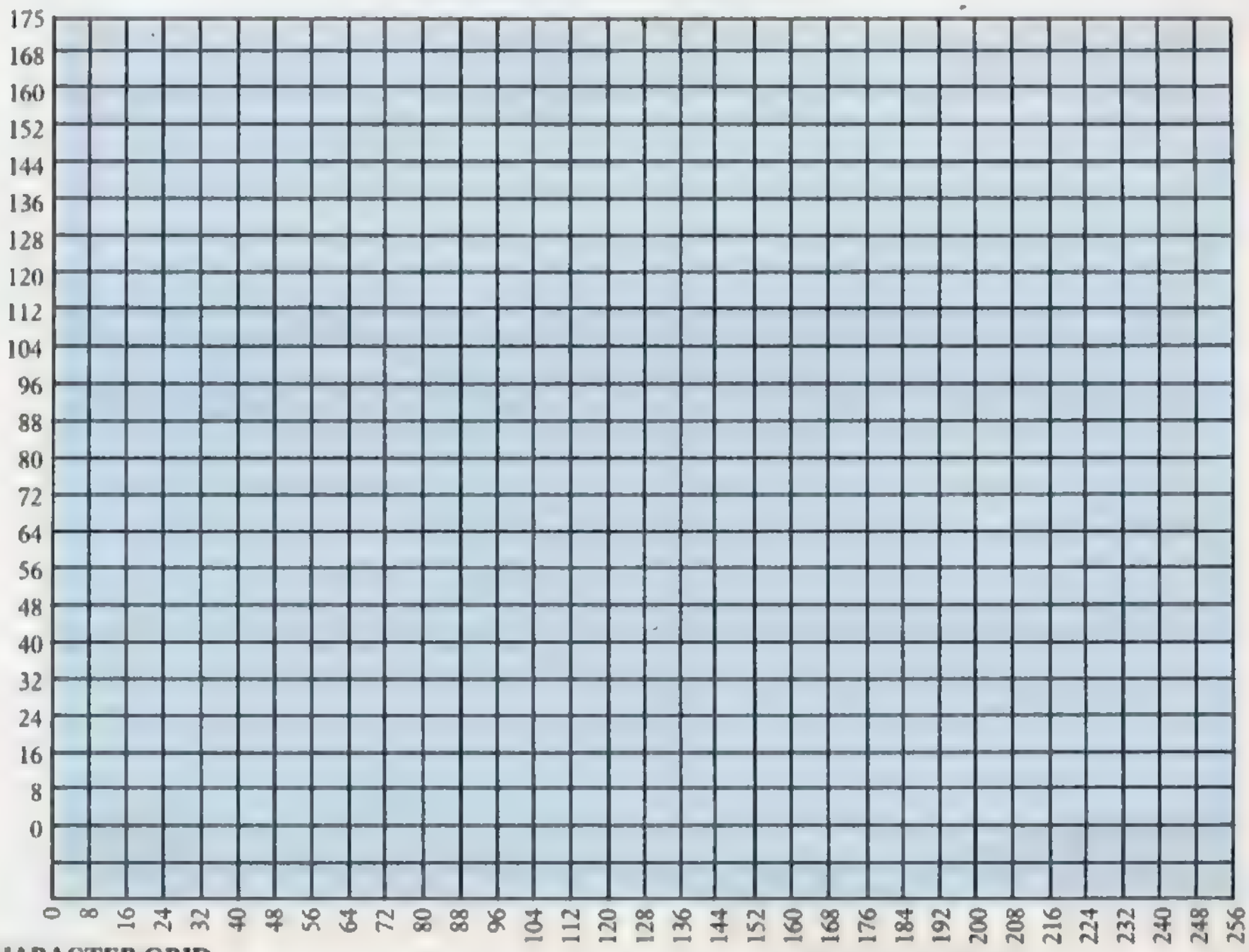
```
LOAD "FOR...NEXT"
```

Make sure the tape is rewound. Start it playing. When the computer finds the program, "program:" followed by the filename will appear on the screen. When the program is fully LOADed, the OK screen prompt reappears. Now you can RUN the program.

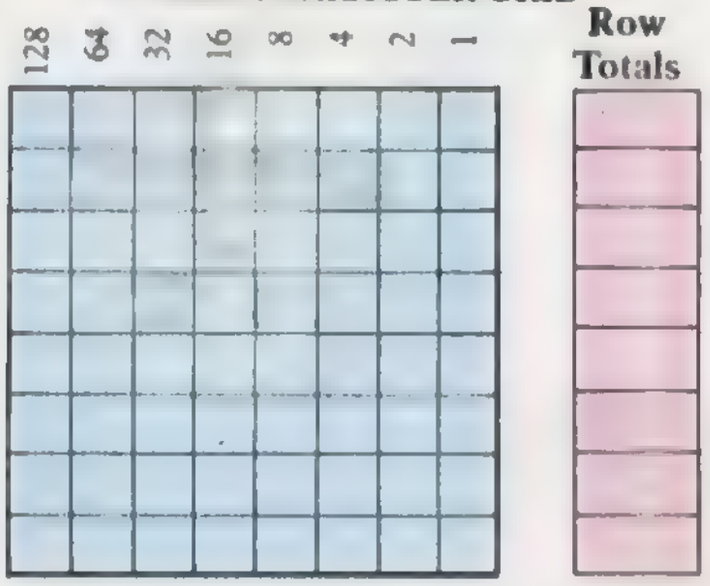
GRAPHICS AND CHARACTER GRIDS

The grid below shows the co-ordinates of the screen display when graphics commands are used. A point on the screen is identified by two co-ordinates x,y. The first co-ordinate sets the horizontal position which is measured along from the left-hand side of the screen. The second co-ordinate sets the vertical position measured from the bottom of the screen. A character PRINTed on the screen occupies an area that is 8 graphics units wide and 8 graphic units high. You cannot PRINT on the bottom two lines of the screen.

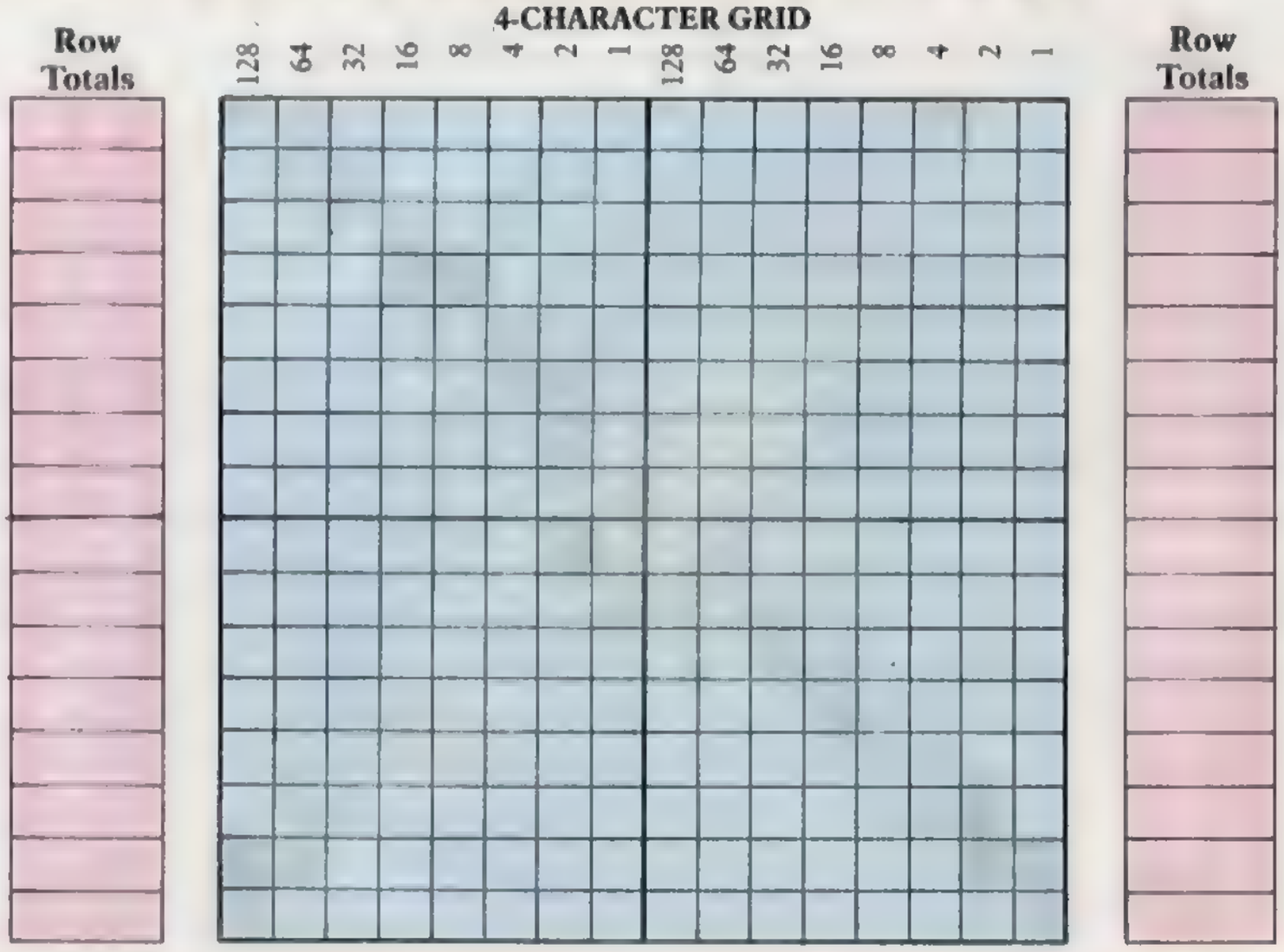
GRAPHICS CO-ORDINATES GRID



SINGLE CHARACTER GRID



4-CHARACTER GRID



Character grids These grids can be used to design either a single character *(above)* or a symbol made from up to four characters *(right)*. You can pencil in your design on the grids and then use the blue columns to list the row totals. These are used in a program with the commands POKE USR. Keys A to U are free for programming user-defined characters.

GLOSSARY

Entries in **bold type** are BASIC keywords.

AT
Used with **PRINT** to place characters on the screen.

BASIC
Beginners' All-purpose Symbolic Instruction Code; the most commonly used high-level programming language.

BEEP
Makes the computer sound a short note or beep, whose duration and pitch are determined by the numbers following the command.

BIN
Converts a number written in binary into the equivalent number written in decimal.

Bit
A binary digit – either 0 or 1.

BORDER
Changes the colour of the screen's border area.

BRIGHT
Turns specified characters to a brighter shade of their **INK** colour.

Byte
A group of eight bits.

Chip
A single package containing a complete electronic circuit. Also called an integrated circuit (IC).

CLS
Clears the text area of the screen.

CPU
Central Processing Unit. Normally contained in a single chip called a microprocessor, this carries out the computer's arithmetic and controls operations in the rest of the computer.

Cursor
A flashing symbol on the screen, showing where the next character will appear.

DATA
Stores a line of data. The computer treats whatever follows **DATA** as information that may be needed later in the program. Used in conjunction with **READ**.

DRAW
Draws a line in the current **INK** colour from the graphics origin at 0,0 or the last point visited to a point specified.

FLASH
Makes characters flash on the screen.

Flowchart
A diagrammatic representation of the steps necessary to solve a problem.

FOR...NEXT
A loop which repeats a sequence of program statements a specified number of times.

GOSUB
Makes the program jump to a subroutine beginning at the line number following the command. The subroutine must always be terminated by **RETURN**.

GOTO
Makes a program jump to the line number following the command.

Hardware
The physical machinery of a computer system, as distinct from the programs (software).

IF...THEN
Prompts the computer to take a particular course of action only if the condition specified is detected.

INK
Changes the colour of text and graphics that appear on the screen.

INPUT
Instructs the computer to wait for some data from the keyboard which is then used in a program.

INT
Converts a number with a decimal fraction into a whole number.

Interface
The hardware and software connection between a computer and another piece of equipment.

INVERSE
Switches the **PAPER** colour for the **INK** colour and vice versa.

K
Abbreviation of kilobyte (1024 bytes).

LET
Assigns a value to a variable.

LIST
Makes the computer display the program currently in its memory.

LOAD
Transfers a program from a cassette tape into the computer's memory.

Loop
A sequence of program statements which is executed repeatedly or until a specified condition is met.

NEW
Removes a program from the computer's memory so that a new program can be keyed in.

OVER
Allows new characters to be **PRINT**ed on top of existing characters without erasing the existing characters.

PAPER
Changes the screen's background colour.

PAUSE
Halts a program for a period set by a number measured in fiftieths of a second.

PLOT
Makes a single dot appear on the screen at the point specified by the co-ordinates that follow it.

POKE USR
Stores a number that reprograms a key to produce a user-defined character.

PRINT
Makes whatever follows appear on the screen.

RAM
Random Access Memory (volatile memory). A memory whose contents are erased when the power is switched off. See also ROM.

READ
Instructs the computer to take a specific number of items from a **DATA** statement so that they can be used in a program.

REM
Enables the programmer to add remarks to a program. The computer ignores whatever follows **REM** in a program statement.

RESTORE
Resets the point from which **DATA** items are **READ**, so that items can be used more than once in a program.

RETURN
Terminates a subroutine. (See also **GOSUB**).

RND
Produces numbers between 0 and 1 at random which can be used to produce unpredictable sequences.

ROM
Read Only Memory (non-volatile memory). A memory which is programmed permanently by the manufacturer and whose contents can only be read by the user's computer.

SAVE
Records a program currently in the computer's memory onto a tape cassette. The program is identified by a filename.

Software
Computer programs.

SQR
Produces the square root of the number that follows it.

STEP
Sets the step size in a **FOR...NEXT** loop.

STOP
Halts a program and **PRINT**s out the line number in which it appears.

String
A sequence of characters treated as a single item – someone's name, for instance.

Subroutine
A part of a program that can be called when necessary, to produce a particular display or carry out a number of calculations repeatedly for example.

TAB
Used with **PRINT** to specify how far along a line characters are to appear.

Variable
A labelled slot in the computer's memory in which information can be stored and retrieved later in a program.

VERIFY
Checks that a program that is currently in memory has been recorded correctly on a tape cassette using **SAVE**.

INDEX

Main entries are in **bold type**

Animation **32–3**, **36–9**, 54–5, 57
Arrow keys 16–7, 22
AT **15**, 25, 62

BASIC 6, **18**, 22, 28, 62
BEEP 10, **42–5**, 62
Binary code/BIN **8**, **53**, 62
Bit **8**, 62
BORDER 10, **34–7**, 62
BREAK 11,23, 26, 59, 60
BRIGHT **39**, 62
Bug *see* Debugging
Byte **8**, 62

Cable 12, 13
CAPS SHIFT key **10–11**, 29, 59, 60
Cassette tape recorder 6, 7, 9, 13, **60**, 63
Character 6, **10–11**, 15, **40–1**, 62, 63
– graphics 52–3
– grids **30–1**, **52–3**, 61
– user-defined **31–3**, **35**, 37, 52, 54, 61
Chip **8–9**, 62
Circuit board 6, **8–9**
Clock, internal 8
CLS **14**, 34, 62
Colour 6, 12–3, **34–9**, **54–5**, 62
– keys 10–11
Connecting up 6–7, **12–13**, **60**
Connectors **6–7**, 8, 13
CPU (Central Processing Unit) **8–9**, 62
Cursor **11**, **22**, 29, 37, 38, 62

DATA **50–1**, 52–3, 54, 62, 63
Debugging 18, **22–3**, 56, **58–9**
DELETE 11, 22
DRAW **28–9**, **38–9**, 47, **54–5**, 62

E (Exponent) **16–17**, 48
EDIT 10, **22**, **59**
ENTER 10, **11**, 12, 14, 18, 22–3
Error message 17, **23**, 59

Fields 15
Filename 60, 63
FLASH **40–1**, 63
Flowchart **19**, 62
FOR … NEXT **26–7**, 33, 44, 46, 52–3, 58, 62, 63

GOSUB **56–7**, 62
GOTO **21**, 26, 44, 56, 58, 62
Graphics 10, 23, **28–33**, 36–9, 47, 49, **54–5**, 61, 62
– characters 29, **30–31**, 41, 54, 61
– grid **28**, **30–1**, 38–9, 52–3, **61**
– key 10–11

Hardware **6–13**, 60, 62

IF … THEN **46–7**, 62
INK 10, **34–5**, 38, 40–1, 49, 54–5, 62
INPUT 23, **24–5**, 27, 62
INT **27**, 48, 62
Interfacing **6–7**, 13, 60, 62
INVERSE **38–9**, 62

Joystick 6, 7

K (Kilobytes) 6, 8, 62
Keyboard **6–7**, 8, **10–11**, 23, 59

LET **15**, 63
Line-numbering 18–19, 22
LIST **20**, 59, 63
LOAD 23, **60**, 63
Loops 19, **26–7**, 44–5, 46, 52–3, **56–7**, 58–9, 62, 63
Loudspeaker 6, **9**, 12

Machine code 8
Mathematical symbols 15, **16**, **46–7**
Memory 6, **8–9**, 18, 20, 23, 56, 60, 63
Menu 57
Microdrives 6, 7, 13
Modes 10, 11
Modules 36, 56

NEW 8, **21**, 63
Number keys **10–11**, 45
Numbers *see* Variable

OVER **38–9**, 63

PAL encoder 8
PAPER 10–11, **34–5**, 36–41, 55, 62, 63
PAUSE **27**, 33, 63
Peripherals **6–7**, 13
Perspective 54
Pixels 28
PLAY 60
PLOT **28–9**, **38–9**, **54**, 63
POKE 59
POKE USR **30–1**, 37, 61, 63
Power supply 6, 8, 9, 12, 13, 18
PRINT 10, **14**, 16, 18–19, 23, 31, 34, 38, 40, 62, 63
Printer 6, 7, 13, 58
Punctutation 15, 17, **19**, 22, 23

READ 37, **50–1**, 52, 62, 63
RECORD 60
REM **18**, **58**, 59, 63
RESTORE **51**, 63
RETURN **56–7**, 62, 63
RND 45, 46, **48–9**, 59, 63
RUN **18–19**, 21, 23, 58, 59, 63

SAVE **60**, 63
Scroll 21, 23, 36, 59
Setting-up **12–13**, **60**
Socket 6, 7, 9, 60
Software 63
Sound 6, 8, 10, 12, **42–5**, 59, 62
SPACE **11**, 23
Speed 27, 33, 42–3, 56
SQR (Square root) **16–17**, 23, 63
STEP 63
STOP 23, 63
String *see* Variable
Subroutine **56–7**, 62, 63
SYMBOL SHIFT key 10–11

TAB **15**, 63
Time delay 33, 45
Tuning-in 12, 35, 60
TV receiver 6–7, **12**, 13, 35

Variable **14–15**, 19, 23, 24–5, 50–51, 59, 63
VERIFY **60**, 63
Video monitor 12–13

DK

ScreenShot

PROGRAMMING SERIES

The original and exciting new teach-yourself programming course for ZX Spectrum owners.

Over 150 unique screen-shot photographs of program listings *and* programs in action – showing on the page exactly what appears on the screen.

Packed full of programming tips and techniques, reference charts and tables, and advice on how to get the most out of your ZX Spectrum.

CONTENTS INCLUDE

Setting up and starting off • Inside your computer • Screen layout and how to control it • Computer conversations • The electronic drawing-board • DIY graphics • Animation • Special effects • Compiling a data bank

Further volumes in the *ScreenShot* series include

Step-by-Step Programming for the **ZX Spectrum** BOOK TWO

PLUS

Step-by-Step Programming for the **BBC Micro, Commodore 64, Acorn Electron,** and **Apple II**

DORLING KINDERSLEY

ISBN 0-86318-026-4

00595

£5.95

9 780863 180262